THE LOGICAL PRINCIPLES OF SOME SIMPLE COMPUTERS

ACADEMISCH PROEFSCHRIFT

TER VERKRIJGING VAN DE GRAAD VAN DOCTOR IN DE WIS- EN NATUURKUNDE AAN DE UNIVERSITEIT VAN AMSTERDAM, OP GEZAG VAN DE RECTOR MAGNIFICUS DR M. W. WOERDEMAN, HOOGLERAAR IN DE FACULTEIT DER GENEESKUNDE, IN HET OPENBAAR TE VERDEDIGEN IN DE AULA DER UNIVERSITEIT OP WOENSDAG 1 FEBRUARI 1956 DES NAMIDDAGS TE 4 UUR

DOOR

WILLEM LOUIS VAN DER POEL
GEBOREN TE 'S-GRAVENHAGE

UITGEVERIJ „EXCELSIOR" - ORANJEPLEIN 96 - 'S-GRAVENHAGE

Promotor: Prof. dr ir A. van Wijngaarden

C O N T E N T S

0. INTRODUCTION

It is the purpose of this thesis to give a number of designs for the logical construction of an automatic computing machine. They all will be based on the same principles.

The paper consists of three parts. Part 1 describes the logical system of an existing computing machine, called PTERA. A new project for a machine, called ZEBRA, was obtained by elaborating the fundamental idea of PTERA, and by removing some objections inherent in PTERA, while, moreover, the logical structure was simplified as much as possible. The logical system of ZEBRA will be discussed in Part 2.

The structure of both designs gave rise to some theoretical problems concerning the greatest possible simplicity that can be realised. These problems will be discussed in Part 3, and they will be elucidated by means of the design of a machine (called machine ZERO) discussed by the author in an earlier publication and by the designs of some structures which are even simpler.*)

For these designs the corresponding systems of conventions for programming are of great importance. This holds more for ZEBRA than for PTERA, because in the case of the former a simpler design was aimed at, which involved, however, more difficult programming.

All the machines operate in the purely binary system.

Computing machines may be divided into two main groups:

1. Machines which store their instructions and the numbers on which they operate in separate parts of the store, and often also put them into the store along separate paths.
2. Machines in which instructions and numbers are stored in the same store.

Machines of the first category are mostly equipped with a unit by means of which the course of the programme can be influenced by certain numbers. Thus it is possible to make the machines of the first category perform everything which machines of the second category perform (see Part 3). Machines of the second category are able to calculate with their own instructions and are much more flexible. This thesis will deal with machines of the second category only. The two categories cannot be clearly distinguished from one another, because there are also mixed forms.

The parts of every automatic computer are:

1. Store
2. Arithmetic unit
3. Control
4. Input unit(s)
5. Output unit(s)

---

*) W. L. v. d. Poel. A simple electronic digital computer. Appl. Sci. Res., B2(1952)367.

1. The function of the store is the storing of data and intermediate results of a computation. These data are not only numbers but also instructions indicating what must be done.

The store may be envisaged to be divided into several numbered boxes, each of which can contain a word (number or instruction). These boxes will be called registers, and their reference numbers will be called addresses.

The registers can be situated either in space (all of them being accessible at the same time), or in time sequence (an access time being required to read a number in or out). These forms are called parallel and serial storage respectively. The digits within a word can also be in parallel or in series. Often a store is operating partly in parallel and partly in series.

A machine can be provided with some type of storage having a different feature, e.g. a large but slow store backing up a small but high speed store.

For the sake of economy a magnetic drum has been chosen as storage for PTERA and ZEBRA. For a large and moderately quick store this is the cheapest solution at the moment. In both cases a serial representation of the numbers will be made use of. 32 numbers are placed on a circumference. Several (32 and 256 respectively) tracks are available in parallel, only one track being used at a time.

The principles given can be applied, without further considerations, to stores with comparable features, such as electric or acoustic delay lines.

In ZEBRA a small but high speed store, consisting of delay lines, is used besides the drum store.

2. The arithmetic unit is the part in which are performed the actual arithmetic operations such as addition, multiplication, etc., and also logical operations like conjunction and shifting.

The arithmetic unit in most cases consists of one or more registers destined to accumulate sums. These registers will be called accumulators.

It is not necessary that the arithmetic unit should have the facility of carrying out all the arithmetic operations. PTERA has been provided with multiplication and division facilities, which is not the case with ZEBRA. By means of programming these operations can be built up from their more elementary parts. In Part 3 it will be investigated what functions the arithmetic unit should at least be able to perform.

3. The control co-ordinates the functions of the arithmetic unit and the store, and it extracts the instructions from the store in the right sequence.

In the machines to be discussed a great part of the control operations is often performed in the arithmetic unit. Conversely the control can be provided with an adder which enables it to compute with instructions, so that especially with regard to the machines to be dealt with here, it is not possible to make a strict discrimination between control and arithmetic unit. Often they will together be designated by "operational part of the machine".

It is just the structure of the control which has the essential characteristics that distinguish the machines to be discussed from other machines. Normally the following phases can be distinguished:

a. the extraction of the instruction from the store;
b. the taking over of this instruction, setting up the correct route for the words in executing this instruction;
c. the extraction or storing of the operand and the performing of the operation;
d. the taking over of the address from which the next instruction must be extracted.

Mostly the phases a and c require a whole word time and the actions b and d are performed in the interval between the words, so that if the word time is considered to be the elementary period, the phases a and b together form the 1st period, and the phases c and d together the 2nd period. This first period will be called the instruction period and the second period will be called the operation period.

As to the machines dealt with in this thesis it will appear that no essential difference between the two said periods exists. Often in PTERA and ZEBRA these periods can be distinguished to a certain extent, but there is no rigid alternation of operation and instruction period. Moreover, the two periods are treated technically in perfectly the same manner and in the same unit. In ZEBRA the two periods are even perfectly identical. Instruction and operation period have a dual character.

An instruction is composed of two parts, a functional part, briefly called operation, and an address part containing one or more addresses. The addresses designate the locations of the operands, and the operation gives the task to be performed on these operands. Sometimes an operation can be given implicitly by means of an address with a special function.

According to the number of addresses applied, one-address, two-address, three-address, and four-address codes and even more-address codes can be distinguished.

The address in the one-address machine can only designate one operand, the other operand is situated in an accumulator. The instructions to be executed in succession are often situated at sequential places of the store. PTERA and ZERO are purely one-address machine.

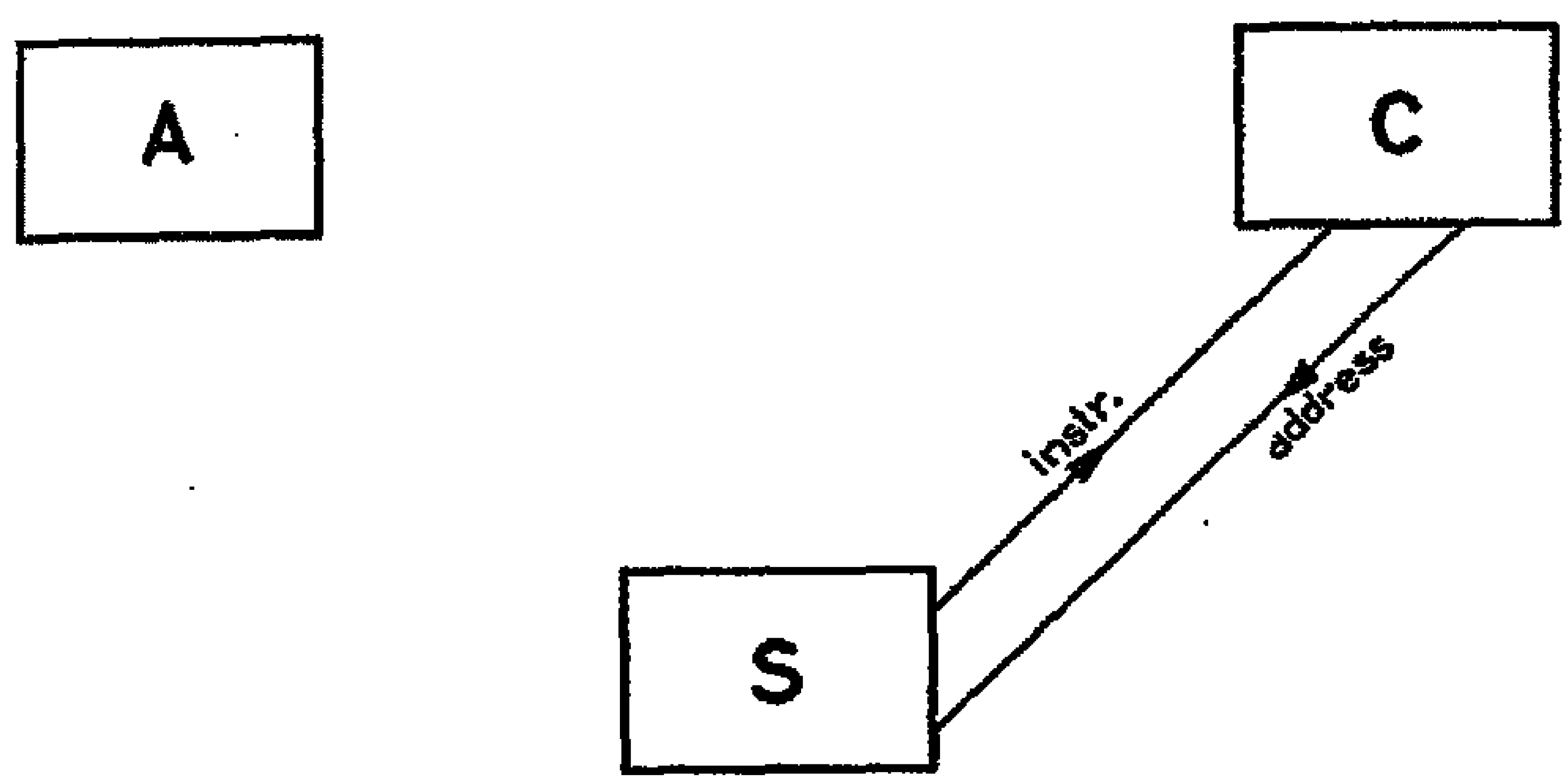
In ZEBRA the two-address code is applied. The functional character is dependent on the instruction used and can be the same as the character of a one-address machine, but it is also possible to make use of one or both addresses to indicate the location of the next instruction.

Three-address machines use either all three addresses for an operation in the form A op B ⟶ C and then have the instructions in normal sequence, or they use the third address to indicate the location of the following instruction.

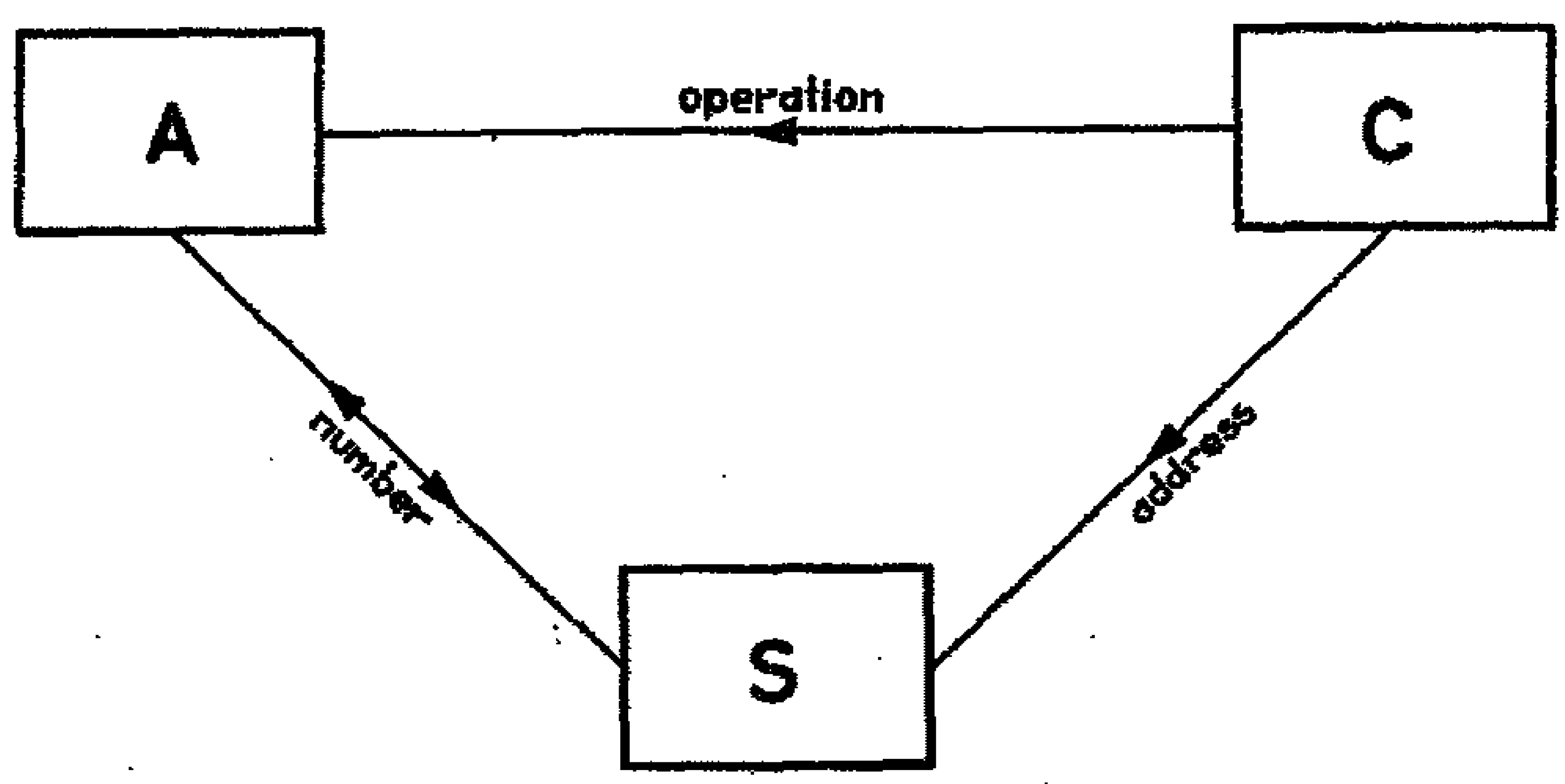
Four-address machines often also have an operation of the form A op B ⟶ C and use the fourth address for the designation of the location of the next instruction.

In PTERA the control unit is treated in the following special manner.

If we indicate the control by C, the arithmetic unit by A, and the store by S, the following will take place during the instruction period:

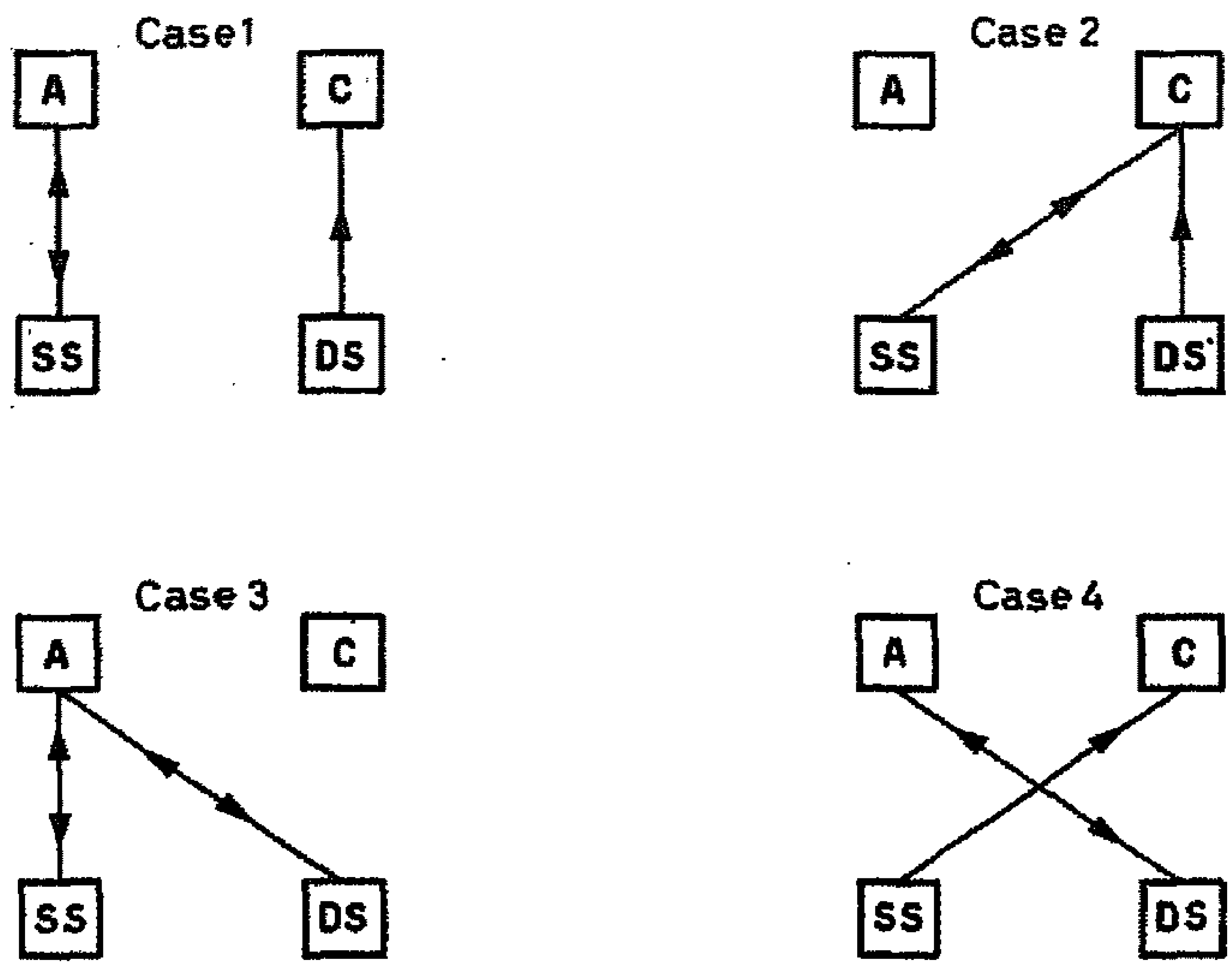


and during the operation period:



The first figure shows that the arithmetic unit is not used for computing. The address from which the instruction has been extracted, must, however, be increased by one to extract the next instruction from the next address. In PTERA and ZERO the same arithmetic unit is used for the adding procedure.

ZEBRA has two different forms of storage, a small but high speed store (called short store) and a large but, as regards average access time, slower drum store. If the drum store is indicated by DS and the short store by SS, both addresses (a short address and a drum address) can be used in the following four manners:



Case 1 is called adding jump. (The transition A ⟶ SS need not be of actual importance). Operation period and instruction period coincide here.
In case 2 both addresses are used for the control. This is called a double jump.
In case 3 both addresses are used for the arithmetic unit. This is called a double addition.
Case 4 is called the jumping addition and can be used together with case 2 to make automatically variable instructions (B-tube facility, modification).

It is a common feature of all systems that also the extraction of an instruction in itself is an instruction with an address and an operation part. This operation part is maintained when the address is advanced by unity and especially in ZEBRA it has a very important function. For this reason the instruction cycles have as important an arithmetic function as the pure operation cycles.

4. The task of the input unit is the feeding of data into the machine. Here standard teletype punched-tape will always be used as a medium. The tape may be considered as an auxiliary store with immense capacity.

5. The output unit is used to take the results out of the machine and put them on a medium such as paper or magnetic tape. With PTERA use has been made of a modified electric typewriter. For ZEBRA a normal teleprinter will be applied. The output may be either typed out or punched on a tape.

Part 1. PTERA

1.1 Considerations concerning the selection of parts for PTERA

When the construction of a computer is started, its character will for the greater part be determined by the kind of store and the components to be used. The choice of these details is mostly dependent on economical factors. Among the various types of stores, the magnetic drum is outstanding for its simplicity and low price, and that is why this kind of store has been chosen.

It is possible to extract the digits in parallel from the various tracks of the drum, but this procedure requires in any case separate writing and reading apparatus for each track, which is rather expensive. Moreover, the time required to find successively an instruction and a number at an arbitrary place, will yet be the time required for one revolution. (An improvement may be obtained through optimum programming, but then a number of high speed registers must be available). It will therefore be simpler to store the numbers serially. Then there will always be only one track operating at a time, so that a common reading and writing apparatus suffices.

The character of the store makes it very suitable for the recording of information in binary form, which in itself need not be an impediment to making a decimal machine, as use can be made, if desired, of a binary coding for the separate digits. It is, however, not to be denied that the arithmetic unit, and more particularly the multiplication and division units will become more complicated. For a machine which will be used mainly for scientific purposes, the number of computations will as a rule be large with respect to the number of data which have to be treated, such in contrast with machines for commercial applications, which have to perform relatively few operations on many data. Furthermore in the Central Laboratory of the Netherlands Postal and Telecommunications Services there had already been developed a binary adding and multiplying arrangement containing only $4\frac{1}{2}$ valves plus 1 relay per digit place. Therefore it was decided to make the machine purely binary.

The first question to be faced, if the binary system is used, is: in what manner must negative numbers be represented inside the machine? This can be effectuated by storing the number in the store as an absolute value and sign, or it can be done in complement form *). As, normally, the numbers inside the machine need not be considered by the operator, the use of

---

*) If a number p consists of the digits $p_o, p_1, \ldots\ldots\ldots, p_n$, and the point is between p_o and p_1,

$p = (1 - 2p_o)\sum_{j=1}^{n} p_j 2^{-j}$ is called the modulus & sign system

$p = -p_o + \sum_{j=1}^{n} p_j 2^{-j}$ is called the complement system

$p = \sum_{j=1}^{n} (p_j - p_o) 2^{-j}$ is called the pseudo-complement or inverse system the complement form also in the store is certainly preferable, the more so as in that case a uniform process for adding and subtracting can be applied, it being able to treat the sign digit in the same way as the other digits.

There is still a possibility to choose between the pure complement form and the pseudo-complement form. In machines in which only operations in series are carried out, the pure complement system can best be used because it does not require a carry-around. (See parts 2 and 3).

For machines which nevertheless require a parallel adder for the multiplying unit it is recommendable to apply the pseudo-complement or inverse system, because it has the facility that a number can very easily be made negative (replacement of 0's by 1's, and conversely) so that many instructions can have an additive as well as a subtractive variant. That is why the pseudo-complement system was chosen for PTERA.

The number of binary digits in a word is determined by the character of the computations occurring normally. Because of the fact that 10^9 is suitable and just a little smaller than 2^{30}, a word length was chosen of 30 binary digits plus sign digit. Apart from that, this choice is rather arbitrary.

Experiments showed that the choice of 32 tracks, each of them with 32 numbers, could be realised very well. The number of revolutions chosen is 2400 revolutions per minute; the packing density has been safely chosen at 3 impulses on a mm.

1.2 The principle of the system

As already mentioned, the store has been composed of 32 tracks, each of which containing 32 31-digit words. The distribution of the digits of the words on a track can be arranged in many ways, two of which will be mentioned here, viz.:

1. The digits of the words can be consecutive impulses, so that a whole number becomes available in about 600 μs. There is a waiting time till the number required passes the reading head.
2. Each of the numbers can be distributed on the track in such a manner that first all the first digits of the numbers come out of the store, then all the second digits, etc. So the reading of a number always requires a complete revolution.

Because of the following considerations the second system will be used in the machine:

a. In system 1 the whole number enters in 600 μs, while nothing happens for the greater part of the revolution.

b. In system 2 there is so much time between the consecutive digits of a word that a whole parallel addition can be made in the interval, a thing which would be much more difficult with system 1.

It is a drawback of system 2 that the extraction of an instruction and a number always requires 2 revolutions against 1 revolution in system 1. Moreover, system 1 can be made suitable for optimum programming (See part 2).

1.21 Adding and multiplying

To add a number which comes out of the store in series, to a number which is already present in a register of the arithmetic unit, actually only one adding mechanism is required which can add one digit at a time (if at least the numbers are added from right to left). In order to carry out also a multiplication in one revolution, at least 31 adding mechanisms are required simultaneously to make it possible to add the multiplicand in parallel to the partial result, while the multiplier, coming out of the store in serial form, controls the additions. Once we have got these required 31 adding units, we can as well add two numbers, the most significant digit coming first. So we may still choose between having the numbers in time sequence with the most significant digit as first digit or with the least significant digit as first digit.

When multiplying is carried out from right to left (so when the least significant digit of the multiplier is dealt with first) only an arithmetic unit with 31 adding mechanisms is required to obtain a product of two 31-digit numbers, because, when each partial product is added, we get on the right-hand side a digit which does not change further. It is, however, a drawback that we do not know the sign digit of the multiplier beforehand, as this digit enters only as last digit.

When multiplying is performed from left to right it is possible that, when the last addition of the multiplicand to the partial result of double length has been made, we get a carry of 62 places maximum. So then 62 adding mechanisms are required. It is an advantage that the sign that leaves the store as first digit, is known directly at the beginning of the multiplication, so that it can be taken into account with the treatment of the signs.

It is not objectionable that we may get a carry-over of 31 places maximum, when adding takes place from left to right, because in any case 31 adding mechanisms are available.

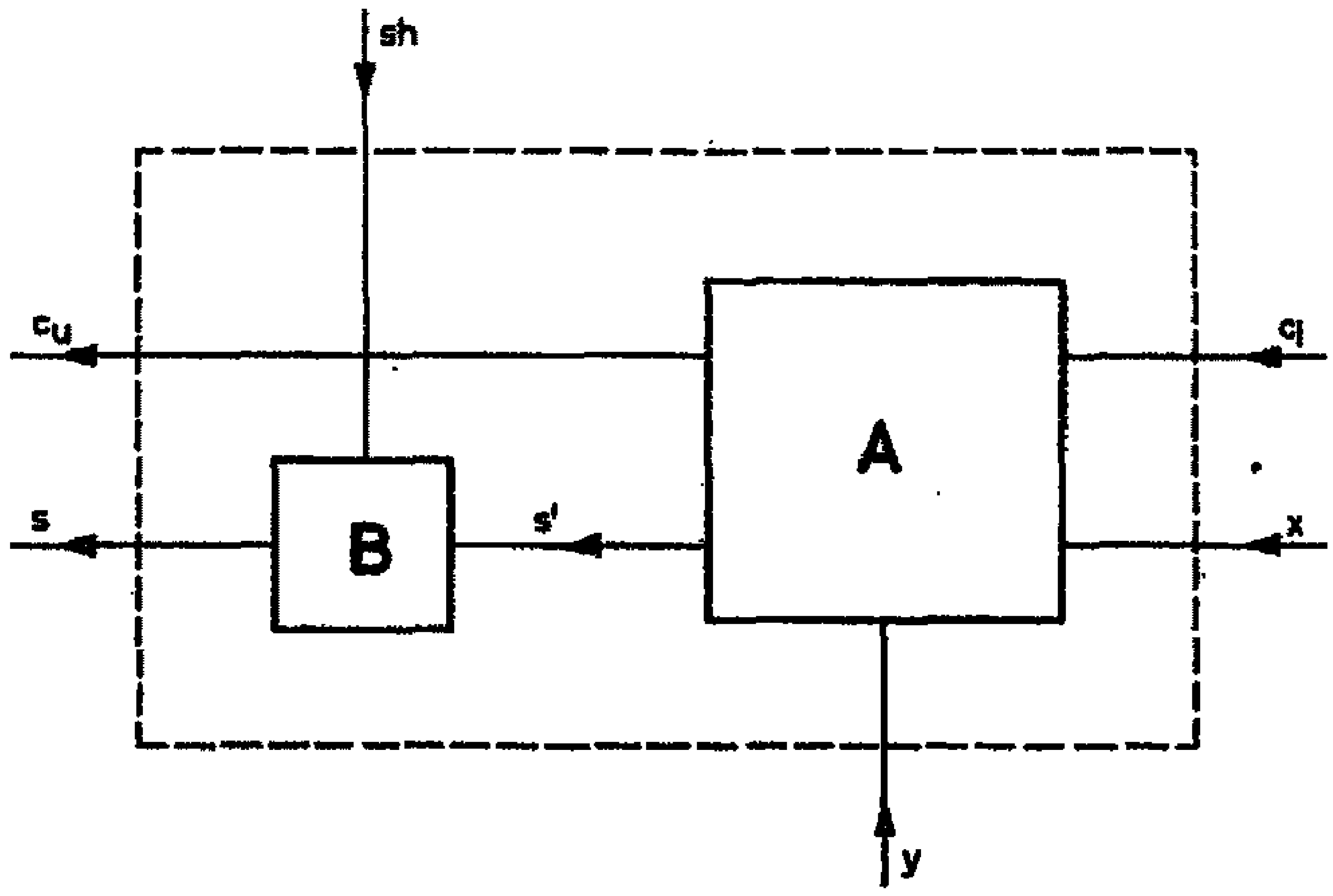
The division process can also be effected in an elegant manner. Division is, however, an operation in which the digits of the quotient are always produced from left to right. Also in connexion with the advantage that the sign digit comes first, the system has been chosen in which the most significant digit comes first, notwithstanding the drawback of two registers having complete adding facilities. Apart from that, this second adding unit plays an active rôle in the control for advancing the address count.

Addition is performed in series. When the digits shift into the arithmetic unit, the most significant digit comes first. When multiplications are made, and the digits of the multiplier leave the store, the most significant digit also comes first; so at each addition the partial result must shift one place to the left in the arithmetic unit. When divisions are made the partial result moves one place to the left between every two additions, while the quotient is produced with the leftmost digit coming first.

From this it appears that an arithmetic unit is required which with every elementary addition (either of a whole number or of one digit at a time) always moves the result one place to the left.

By the application of this rigidly built-in facility of shifting to the left the technical construction is much simplified.

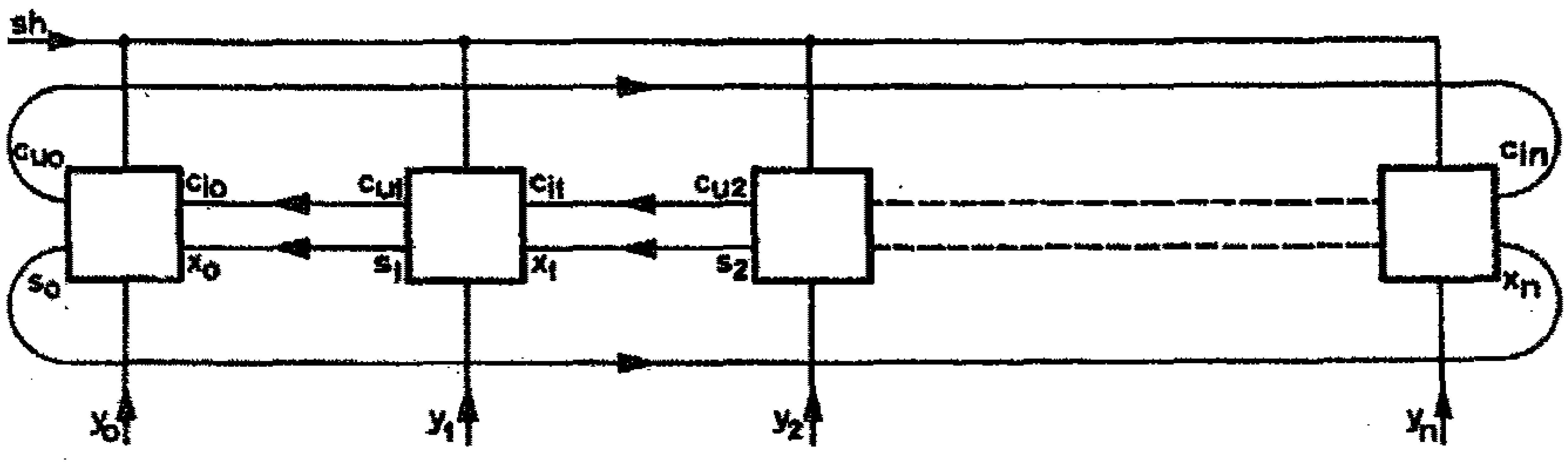
A digit place of the arithmetic unit can be represented as follows:



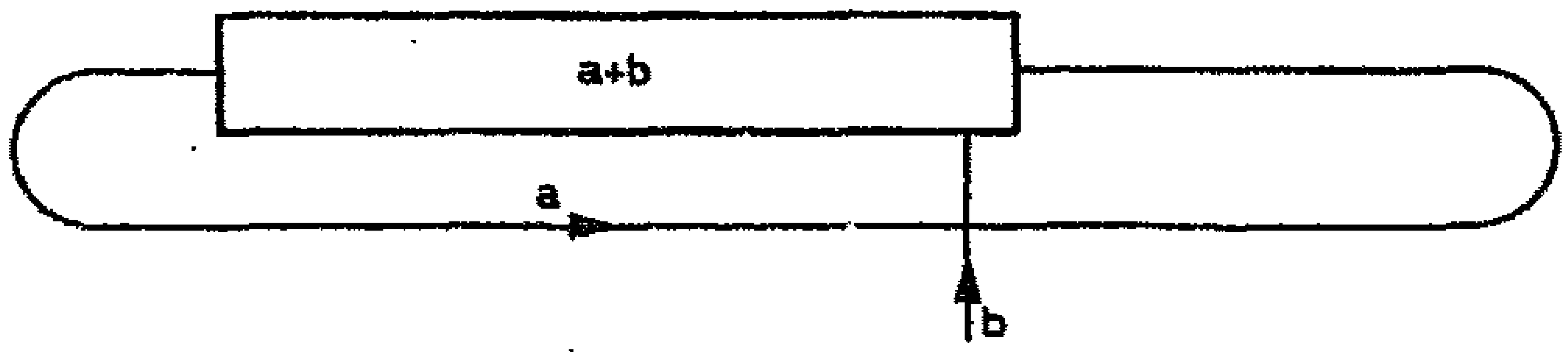
x and y represent the digits of the numbers to be added, c_i is the transfer from the previous place, s' is the digit of the newly formed sum, s is the sum digit recorded on the previous occasion, and c_u is the transfer to the next place.

The whole stage consists of two parts indicated by A and B. A is the actual sum-former which, as quickly as the switching time of the element permits, forms the sum s' as soon as the digits x, y, and c_i have been applied. The part B is a storing element which records an offered digit s' at the moment when the so called shift impulse sh is given. The recorded sum remains statically available at the output.

By interconnecting these elements in the following manner:



an arithmetic unit is obtained, which always shifts and adds simultaneously. In each digit place the digit to be added plus the sum digit of the previous place plus the carry-over from the previous place are added. Then the process of serial adding can be represented by:



(The situation which results is drawn).

The number which is already present in the arithmetic unit is circulating while the digit of the number to be added is being added to it. After the number has circulated, the register will contain the sum. This register is called the accumulator (indicated by A). (In the logical diagrams the registers are always drawn in such a manner that the numbers are present in it as they are written, on paper, so with the least significant digit on the right-hand side).

1.22 The multiplier

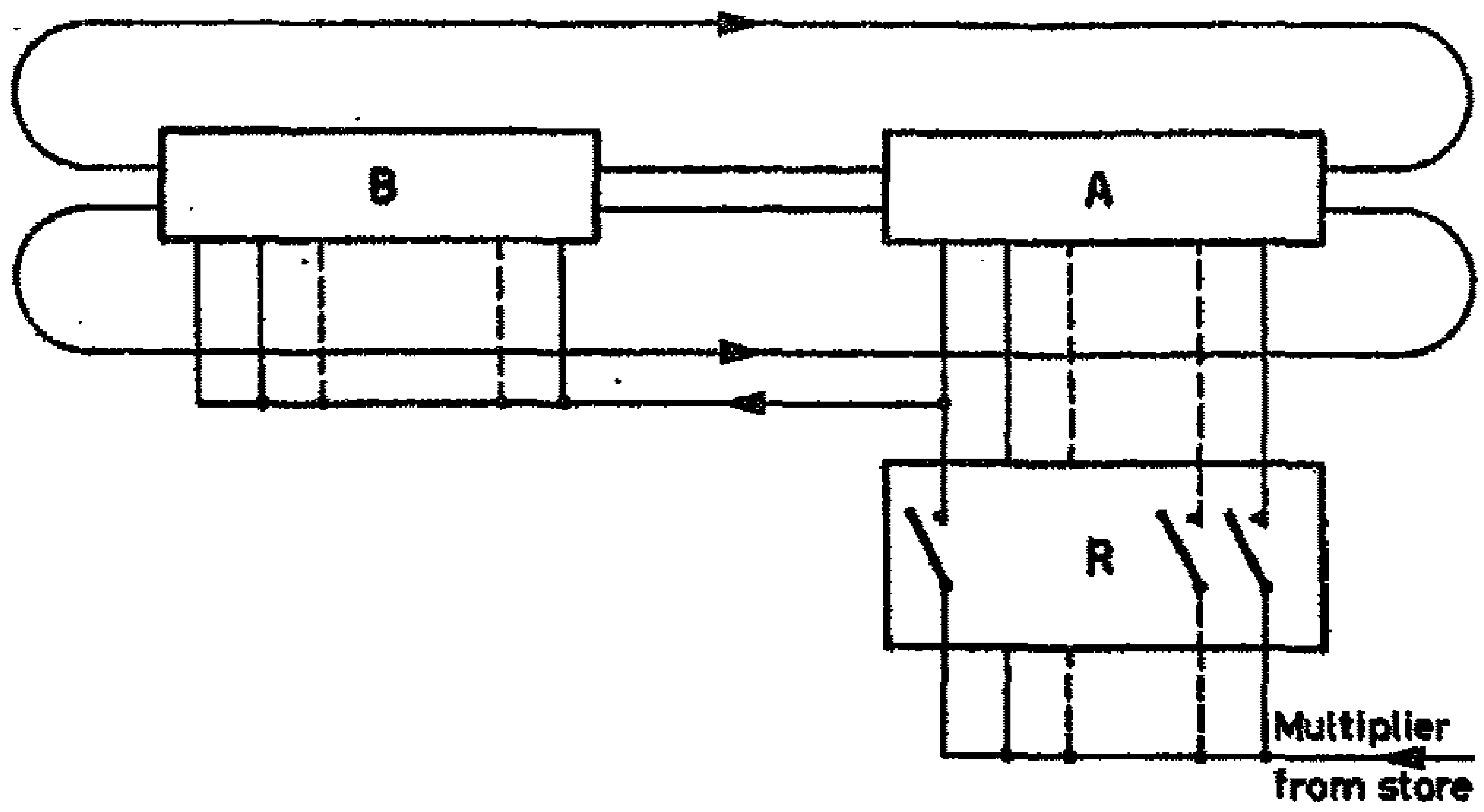
For the multiplying process four registers are fundamentally required: one for the multiplicand, one for the multiplier and two for the product. It has been shown by VON NEUMANN et al. *) that it is also possible to work with three registers, the multiplier and the least significant part of the product having one register in common (See part 2). The two registers for the product will be formed by the double-length accumulator, of which A will be used for the least significant part of the product and the other register B for the most significant part. For the most significant part we shall introduce the term "head" and for the least significant part the term "tail". The register for the multiplier is situated in the store. The digits of the multiplier leave this register sequentially. The register for the multiplicand has been constructed with relays and will be called the relay register (indicated by R). The reasons why no electronic register has been chosen are the following:

1. The register need not be able to compute, and it need to be loaded only once in a revolution. Technically this can be realised in a very simple manner (a half valve plus a relay per digit place).
2. In the non-energised condition a relay of the type applied can remember a digit because it is bi-stable.
3. The relay contact can serve very well as a gate for the digits of the multiplier.

The block schematic diagram for the multiplication is:

---

*) A. W. Burks, H. H. Goldstine, John von Neumann. Preliminary discussion of the logical design of an electronic computing instrument, 2nd Edition. Princeton, N. J. The Institute for Advanced Study, 1947.

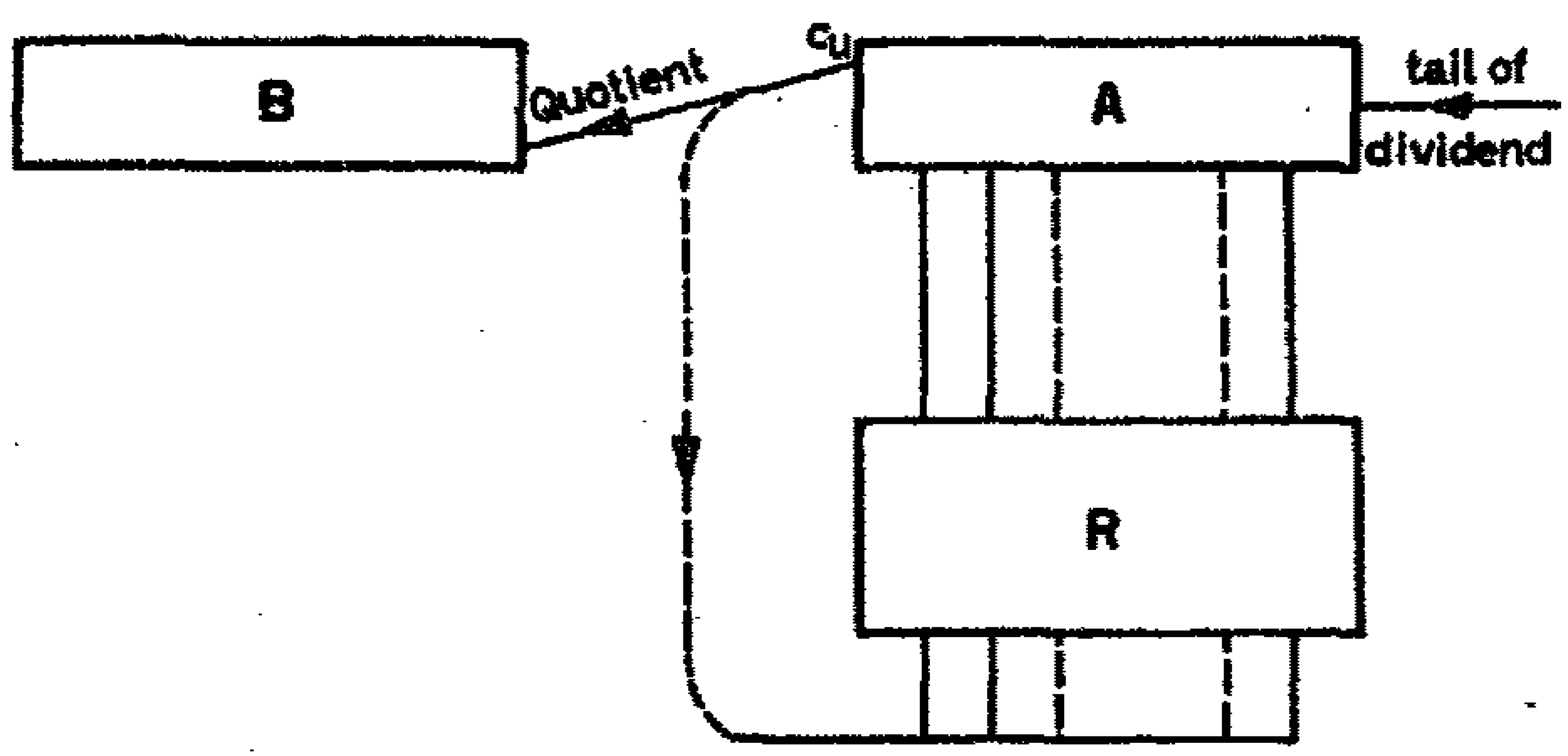


The incoming digits of the multiplier determine whether the multiplicand is either or not added to the double-length A and B registers. The shifting facility of A and B provides for the shifting of the partial result. It is also pointed out that the multiplicand is supplemented on the 31 digit places of the head, to assure that the correct sign digit is formed.

1.23 The divider

The division process is in many respects the inverse operation of the multiplication. So in the division there are also 4 numbers of single length: a double-length dividend, a divisor and a quotient which will be supposed to be of single length. In contrast with the multiplying process the tail of the dividend need not be operated upon; while the division is proceeding, the tail can arrive from the store. The subtractions are always of single length and can be performed in A. The divisor is present in R and the quotient, which is formed just by the carry-overs, with due consideration to the sign rules, can flow into B. The determination of each digit of the quotient is done in two parts, viz. subtraction of the divisor in any case, and the cancelling of this subtraction by adding it back again if the operation in question cannot be performed. The carry-over digits determine whether the divisor can be subtracted or not. The adder is operating fast enough to enable these two additions to be done in the same time in which otherwise one addition takes place.

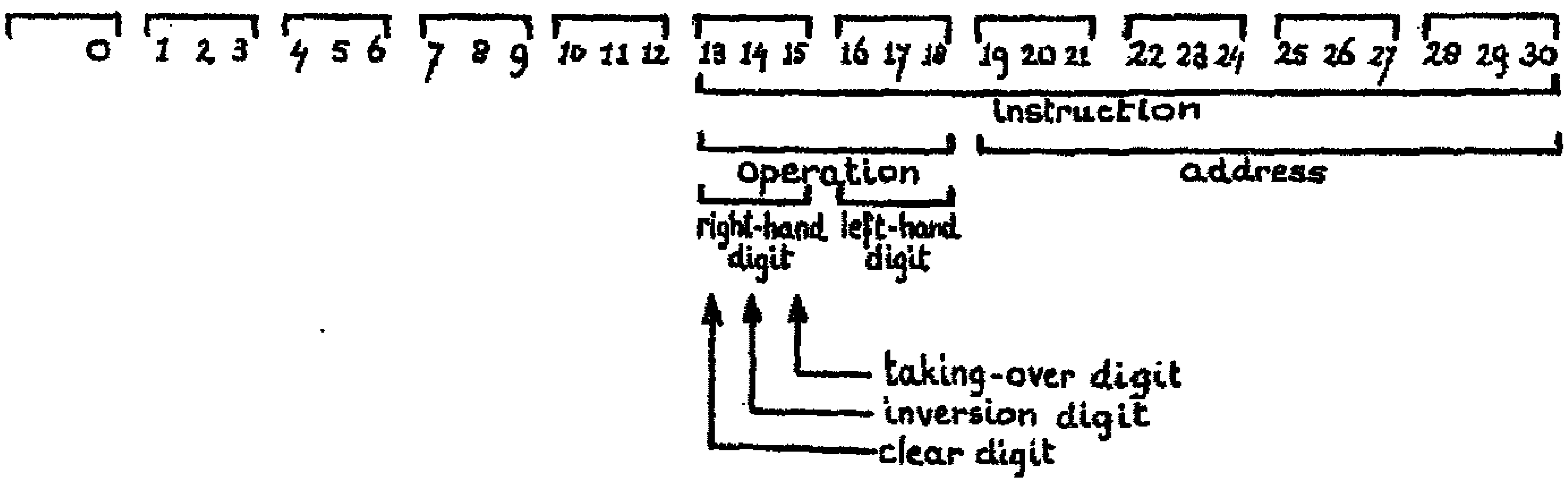
The block schematic diagram of the division process is:



1.3 The systematics of the instructions

One of the fundamental ideas underlying the design of PTERA is the functional use of some digits of the operation. (The same idea is used in ZEBRA in a much more extended form). As a matter of fact the same notions play a rôle in many operations. It is useful to have both an additive and a subtractive variant of each kind of operation. Clearing, too, is an important factor. Either of these two functions has got a functional digit in the operation. As the relay register cannot be filled in a direct manner it was desirable to make the taking-over into the relay register after an instruction dependent on a functional digit. These three binary digits can be expressed in an octal digit, which indicates the variant of the instruction. If we also want to be able to indicate the kind of instruction by an octal digit, we can only indicate eight different instructions in this manner, which is insufficient. It is, however, not necessary to make the functional taking-over digit available for all kinds of instructions. A jump should preferably leave the computing registers intact, and consequently needs never to take over A in R, whereas in the typing process taking-over is constantly necessary for setting up the next digit on the typewriter. Therefore such provisions have been made that for the operations with octal number 0, 1, 2, and 3, giving the kind of operation, the taking-over digit also is of influence on this kind, whereas this is not the case for the species 4, 5, 6, and 7.

The arrangement of the operation and address digits in an instruction is:



The digit place 13 determines whether clearing will take place or not. $c_{13} = 0$ means clearing; $c_{13} = 1$ means non-clearing. The digit place 14 determines whether inverting will take place or not. $c_{14} = 0$ is the additive variant; $c_{14} = 1$ is the subtractive variant. The digit place 15 determines the taking-over. $c_{15} = 0$ means no taking-over, $c_{15} = 1$ means taking over in R. The digit place 16 indicates whether the digit 15 is also of influence on the kind of instruction. If $c_{16} = 0$, c_{15} is of influence, and if $c_{16} = 1$, c_{15} is not of influence on the instruction.

The kinds of instructions are indicated by:

00: stop	40: add	01: input
10: store	50: prepare	11: output
20: jump	60: multiply	21: round off
30: test	70: divide	31: conjunction

So of 40, 50, 60, and 70 all eight variants are present, each of the kinds 00, 10, 20, 30, 01, 11, 21, and 31 having only four variants.

Though in the computer the octal digit determining the species is behind the variant digit, it is written in front in the notation. It would have been possible to choose another solution but for historical reasons this arrangement has been maintained.

1.4 The action of the instructions

The action of the instructions will be described by means of a table indicating the route of the numbers in each operation. This can be done in two ways:

a. A description can be given of the real course of affairs, in which the extraction of an instruction is consequently also considered as a separate instruction.
b. A phenomenological description can be given, in which each instruction is considered as take instruction + do operation. This description need not at all correspond with reality, if only the programmer obtains the right results by means of it. For details cf. *)

We shall only deal with the description under a. The carrying out of each instruction always consists of two phases: the extraction and the taking-over. During the extraction process under the control of the instruction which has already been set up, the word-transport takes place, during the taking-over process the route of the words has to be changed in accordance with the new instruction. The new instruction is always taken over from C to Z, which will not be discussed further. (Cf.**)

In the notation an arrow will indicate: .. goes to register .... The new contents of a register will be indicated by an accent.

The transport to R is always effectuated during the taking-over of the new instruction.

By (n) will be denoted the contents of register n. In the same way (A) will denote the contents of the accumulator A.

Conversely the notation)x(= n will be used when x = (n) .

---

*) W. L. v. d. Poel. Het programmeren voor PTERA. Het PTT-Bedrijf, 5(1953)135.
**) W. L. v. d. Poel. De werking van PTERA. Het PTT-Bedrijf, 5(1953)124.

Table of operations

Stop

n/00 :	$(B) + \varepsilon \to C$	$(A) \to B$	$(n) \to A$	Stop the machine
n/02 :	$(B) + \varepsilon \to C$	$(A) \to B$	$-(n) \to A$	" " "
n/04 :	$(B) + \varepsilon \to C$	$(A)^{*} \to B$	$(A) + (n) \to A$	" " "
n/06 :	$(B) + \varepsilon \to C$	$(A)^{*} \to B$	$(A) - (n) \to A$	" " "

Store

n/10 :	$(B) + \varepsilon \to C$	$(A) \to B$	$(A) \to n$	$+0 \to A$
n/12 :	$(B) + \varepsilon \to C$	$(A) \to B$	$-(A) \to n$	$+0 \to A$
n/14 :	$(B) + \varepsilon \to C$	$(A) \to B$	$(A) \to n$	
n/16 :	$(B) + \varepsilon \to C$	$(A) \to B$	$-(A) \to n$	

Jump

n/20 :	$(C) \to B$	$(n) \to C$	After 20, 30 tests on r_o
n/22 :	$(C) \to B$	$(n) \to C$	
n/24 :	$(C) \to B$	$(n) \to C$	After 24, 30 tests on a_o
n/26 :	$(C) \to B$	$(n) \to C$	
n/28 :	$(C) \to B$	$(\dot{n}) \to C$	After 28, suppress punching

First 20-operation after a 50-operation

n/20 :	$(B) \to A_v$	$(C) \to B$	$(n) \to C$	Switch off extra place
n/22 :	Same as above			
n/28 :				

Second 20-operation after a 50-operation

n/20 :	$(B) \to A$	$(C) \to B$	$(n) \to C$	Switch off the circulation
n/22 :	Same as above			
n/28 :				

Test

With 20 as intermediate jump

n/30 :	$(B) + \varepsilon(1 + r_o) \to C$	$(A) \to B$	$(n) \to A$
n/32 :	$(B) + \varepsilon(2 - r_o) \to C$	$(A) \to B$	$-(n) \to A$
n/34 :	$(B) + \varepsilon(1 + r_o) \to C$	$(A)^{*} \to B$	$(A) + (n) \to A$
n/36 :	$(B) + \varepsilon(2 - r_o) \to C$	$(A)^{*} \to B$	$(A) - (n) \to A$

With 24 as intermediate jump

n/30 :	$(B) + \varepsilon(1 + a_o) \to C$	$(A) \to B$	$(n) \to A$
n/32 :	$(B) + \varepsilon(2 - a_o) \to C$	$(A) \to B$	$-(n) \to A$
n/34 :	$(B) + \varepsilon(1 + a_o) \to C$	$(A)^{*} \to B$	$(A) + (n) \to A$
n/36 :	$(B) + \varepsilon(2 - a_o) \to C$	$(A)^{*} \to B$	$(A) - (n) \to A$

Add

n/40 :	$(B) + \varepsilon \to C$	$(A) \to B$	$(n) \to A$	
n/41 :	$(B) + \varepsilon \to C$	$(A) \to B$	$(n) \to A$	$(A)' \to R$
n/42 :	$(B) + \varepsilon \to C$	$(A) \to B$	$-(n) \to A$	
n/43 :	$(B) + \varepsilon \to C$	$(A) \to B$	$-(n) \to A$	$(A)' \to R$
n/44 :	$(B) + \varepsilon \to C$	$(A)^* \to B$	$(A) + (n) \to A$	
n/45 :	$(B) + \varepsilon \to C$	$(A)^* \to B$	$(A) + (n) \to A$	$(A)' \to R$
n/46 :	$(B) + \varepsilon \to C$	$(A)^* \to B$	$(A) - (n) \to A$	
n/47 :	$(B) + \varepsilon \to C$	$(A)^* \to B$	$(A) - (n) \to A$	$(A)' \to R$

Prepare

n/50 : } Identical with corresponding 40-operations +
n/57 : } switch on circulation and extra place

Multiply (unprepared)

n/60 :	1st stroke	$(B) + \varepsilon \to C$	$0 \to B$	$0 \to A$	
	2nd stroke		$k \to B$	$s \to A$	
	where $k + s.\varepsilon = (n).(R)$				
n/61 :	1st stroke	$(B) + \varepsilon \to C$	$0 \to B$	$0 \to A$	
	2nd stroke		$k \to B$	$s \to A$	$(A)' \to R$
	where $k + s.\varepsilon = (n).(R)$				
n/62 :	1st stroke	$(B) + \varepsilon \to C$	$0 \to B$	$0 \to A$	
	2nd stroke		$k \to B$	$s \to A$	
	where $k + s.\varepsilon = -(n).(R)$				
n/63 :	1st stroke	$(B) + \varepsilon \to C$	$0 \to B$	$0 \to A$	
	2nd stroke		$k \to B$	$s \to A$	$(A)' \to R$
	where $k + s.\varepsilon = -(n).(R)$				
n/64 :	1st stroke	$(B) + \varepsilon \to C$	$(A) \to B$	$\mathrm{sgn}(A) \to A$	
	2nd stroke		$k \to B$	$s \to A$	
	where $k + s.\varepsilon = (A).\varepsilon + (n).(R)$				
n/65 :	1st stroke	$(B) + \varepsilon \to C$	$(A) \to B$	$\mathrm{sgn}(A) \to A$	
	2nd stroke		$k \to B$	$s \to A$	$(A)' \to R$
	where $k + s.\varepsilon = (A).\varepsilon + (n).(R)$				
n/66 :	1st stroke	$(B) + \varepsilon \to C$	$(A) \to B$	$\mathrm{sgn}(A) \to A$	
	2nd stroke		$k \to B$	$s \to A$	
	where $k + s.\varepsilon = (A).\varepsilon - (n).(R)$				
n/67 :	1st stroke	$(B) + \varepsilon \to C$	$(A) \to B$	$\mathrm{sgn}(A) \to A$	
	2nd stroke		$k \to B$	$s \to A$	$(A)' \to R$
	where $k + s.\varepsilon = (A).\varepsilon - (n).(R)$				

On an unprepared 64, 65, 66, and 67 operation the circulation is switched on.

Multiply (prepared with a 50-operation)

n/60 :
n/61 : Identical with corresponding unprepared instructions
n/62 :
n/63 :

n/64 : 1st stroke $(B) + \varepsilon \to C$ $\quad sgn(A_v) \to B \quad (A_v) \to A_v$
2nd stroke $k \to B \quad s \to A$
where $k + s.\varepsilon = (A) + (n).(R)$

n/65 : 1st stroke $(B) + \varepsilon \to C$ $\quad sgn(A_v) \to B \quad (A_v) \to A_v$
2nd stroke $k \to B \quad s \to A \quad (A) \hookrightarrow R$
where $k + s.\varepsilon = (A) + (n).(R)$

n/66 : 1st stroke $(B) + \varepsilon \to C$ $\quad sgn(A_v) \to B \quad (A_v) \to A_v$
2nd stroke $k \to B \quad s \to A$
where $k + s.\varepsilon = (A) - (n).(R)$

n/67 : 1st stroke $(B) + \varepsilon \to C$ $\quad sgn(A_v) \to B \quad (A_v) \to A_v$
2nd stroke $k \to B \quad s \to A \quad (A) \hookrightarrow R$
where $k + s.\varepsilon = (A) - (n).(R)$

Divide

n/70 : $(B) + \varepsilon \to C \quad q \to B \quad r \to A$
n/71 : $(B) + \varepsilon \to C \quad q \to B \quad r \to A \quad (A) \hookrightarrow R$
where $q.(R) + r.\varepsilon = (A) + (n).\varepsilon$
$|r| < |(A)| \quad sgn\ r = sgn(A)$
n/72 : $(B) + \varepsilon \to C \quad q \to B \quad r \to A$
n/73 : $(B) + \varepsilon \to C \quad q \to B \quad r \to A \quad (A) \hookrightarrow R$
where $q.(R) + r.\varepsilon = (A) - (n).\varepsilon$
$|r| < |(A)| \quad sgn\ r = sgn(A)$
The sign of (n) is completely disregarded.

Input

n/01 : $(B) + \varepsilon(1 + u_4) \to C \quad (A) \to B \quad u.(n) \to A \quad (A) \hookrightarrow R$
n/03 : $(B) + \varepsilon(2 - u_4) \to C \quad (A) \to B \quad u.(n) \to A \quad (A) \hookrightarrow R$
n/05 : $(B) + \varepsilon(1 + u_4) \to C \quad (A)^* \to B \quad (A) + u.(n) \to A \quad (A) \hookrightarrow R$
n/07 : $(B) + \varepsilon(2 - u_4) \to C \quad (A)^* \to B \quad (A) + u.(n) \to A \quad (A) \hookrightarrow R$
On switching off the input operation: step tape.

n/11 : Identical with n/61
n/13 : Identical with n/63 and type/punch the four/five most
n/15 : Identical with n/65 significant digits of (B)'
n/17 : Identical with n/67

Round-off

n/21 : $(B) + \varepsilon \to C$	$(A) \to B$	$(r_1 - r_o).(n) \to A$	$(A)' \to R$
n/23 : $(B) + \varepsilon \to C$	$(A) \to B$	$-(r_1 - r_o).(n) \to A$	$(A)' \to R$
n/25 : $(B) + \dot{\varepsilon} \to C$	$(A)^* \to B$	$(A) + (r_1 - r_o).(n) \to A$	$(A)' \to R$
n/27 : $(B) + \varepsilon \to C$	$(A)^* \to B$	$(A) - (r_1 - r_o).(n) \to A$	$(A)' \to R$

Conjunction

n/31 : $(B) + \varepsilon \to C$	$(A) \to B$	conj (n);$(A) \to A$	$(A)' \to R$
n/33 : $(B) + \varepsilon \to C$	$(A) \to B$	conj -(n);$(A) \to A$	$(A)' \to R$
n/35 : $(B) + \varepsilon \to C$	$(A)^* \to B$	(A) + conj (n);$(A) \to A$	$(A)' \to R$
n/37 : $(B) + \varepsilon \to C$	$(A)^* \to B$	(A) + conj -(n);$(A) \to A$	$(A)' \to R$

A result 0 produced by: 04, 06, 34, 36, 44, 45, 46, 47, 54, 55, 56, 57, 05, 07, 25, 27, 35, or 37 is always +0 .

1.41 Addition and jump

The various kinds of operations will not be discussed sequentially, but in groups belonging together.

The operations 10, 20, and 40 will be elucidated by means of a simple programme:

100	200/41	(200) = a
101	37/44	(37) = b
102	37/10	
103	120/20	

The following process will then be effectuated in the registers:

C	B	A	R	
100/20		x	y	
200/41	100/20			
101/20	x	a	a	
37/44	101/20			
102/20	a*	a + b		
37/10	102/20			
103/20	a + b	0		$a + b \rightarrow n$
120/20	103/20			
etc.	120/20	0	a	

As is shown, the instructions contained in the programme alternate in register C with the 20-operation shifting to and fro within the control. Only when there is a jump in the programme the circulating 20-operation is replaced. The normal alternation is then interrupted. This can even happen n times at a stretch if new jumps have constantly to be executed. This interruption of the alternation: 20-operation — other operation is one of the essential characteristics of this kind of machine (Cf. Part 2). It appears from the beforegoing that the instruction cycle must be considered as a normal operation cycle. That is why the address counter does not only contain an address but also an operation part.

This principle can briefly be formulated as follows: the contents of C determine the course of the numbers for the next instruction. The beforegoing implies that there is no longer an essential distinction between an instruction and an operation cycle.

The interruption of the normal alternation by two times (C)→B in succession takes place after every jump. By an artifice also the interruption of the normal alternation in the other direction (by having two times (B)→C in succession) can occur but this is seldom of practical importance (Cf. 1.61 and

1.62). In ZEBRA, however, there exists a complete duality between the two types of interruption.

The following should be elucidated too: the adding of unity to the 20-operation present in B is not effectuated while shifting into the register, but when it shifts out of it to C. The reason is that if the adding process would take place while shifting into the register from C to B, as would have been the normal course of affairs in a serial adding unit, an ε has to be generated somewhere. This ε is the last impulse in a word. So to produce it the number of digits has to be counted off somewhere. It is much easier to produce an impulse on the first than on the last place. That is why an ε is added, while shifting out. This ε is generated as first impulse. As the register always shifts first and then adds, the addition of ε must be effectuated at the y input of the place B_{29}.

In the table A^* indicates a number that need not be equal to A, because the sum shifting to B does not get the right carries from A.

.42 The test instructions

The test instructions 30, 32, 34, and 36 are operating by adding ε or 2ε to (B) when the latter shifts to C. This depends on the sign of R or A. Originally only the testing on R had been provided, but afterwards the need of testing on A was also felt. The character of the intermediate jump determines whether test is made on R or A. As long as a 20-operation is circulating as an intermediate jump, R is being tested; in case of a 24-operation A is being tested.

In order to avoid that an instruction will be lost during tests, the arithmetic action of 30, 32, 34, and 36 is equal to the action of 40, 42, 44, and 46.

.43 Multiplication

When two numbers of single length are multiplied, a product of double length will result. In the machine the multiplicand must always be set up in R and the multiplier must be extracted during the 60-operation. The head of the product is always formed in B and the tail in A. The tail has no independent sign digit but a sign digit which is made equal to the sign digit of the head.

Multiplication is performed in two phases. Technically it appeared not to be possible to add to the number shifting out of B and to use B simultaneously in the double-length accumulator. In the first phase there always takes place $(B) + \varepsilon \longrightarrow C$ and the clearing of the number which is already present in A or the extension of this number to the double length. In the second phase the actual multiplication process is performed.

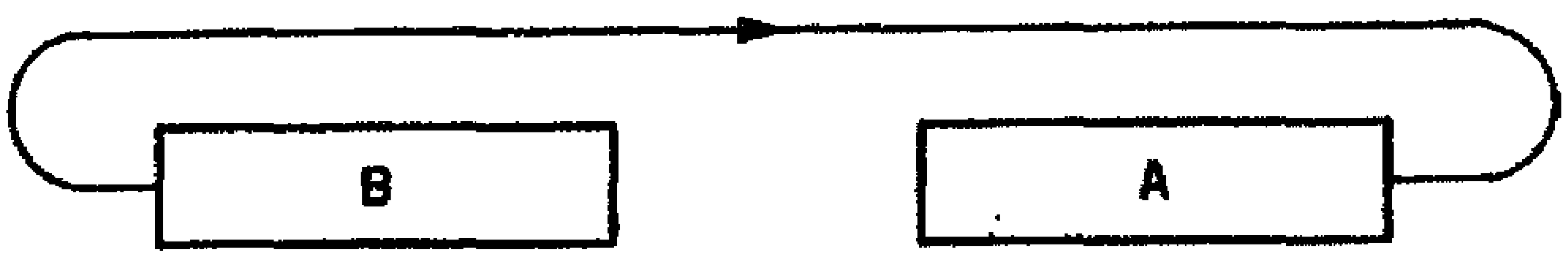
Not always are both tail and head required. If only the tail is required, the non-prepared instructions 60, 61, 62, and 63 are used. The head appearing in B is automatically lost in the subsequent 20-operation. An application frequently used is:

$$)\varepsilon(/60 \ : \ (R) \longrightarrow A$$

$$)\varepsilon(/62 \ : \ -(R) \longrightarrow A$$

It is evident that the head can never be recovered from B to A without further provisions. For this purpose the 50-operations are used.

From an arithmetic point of view the 50-operations perform the same task as the 40-operations but, moreover, they operate two devices, the extra place and the circulation. These devices have the following function: the circulation provides for a connection between B and A so that the contents of B can flow to A during the subsequent 20-operation:



The extra place is a 31st digit place of the accumulator A, a place which has only shifting facilities and no adding facilities. The extra place creates a possibility to set up the number to be added (in a non-clearing multiplication) or the dividend (in a division) shifted back over one place. Then the number is not present in $a_0 - a_{30}$ but in $a_1 - a_{31}$. This is necessary because the construction of the accumulator is such that first shifting takes place and then adding.

The extra place is only required when the multiplication and the division are being prepared; it is switched off after the first 20-operation coming after the 50-operation. The circulation must remain active after the multiplication to get the head of the product from B to A or to obtain the quotient in case of a division process. The circulation is switched off on the second 20-operation coming after the 50-operation.

The 50-operations can be used with the instructions of the 11, 20, 60, or 70 type.

A representative example of the use of the multiplication is:

n	)a(/40
n+1	)b(/51
n+2	)c(/65
n+3	)k(/10

In this example the following activities are performed in the registers:

C	B	A	R	
n/20				
)a(/40	n/20			
n+1/20		a		
)b(/51	n+1/20			
n+2/20	a	b	b	
c/65	n+2/20	a		
n+3/20	sgn a	a		
n+3/20	k	s	s	
)k(/10	n+3/20	k		k → store
n+4/20				

In this multiplication a is added to the head of the product of double length. The tail can be taken over in R and can be brought back into A by)ε(/60 or can be used directly for rounding off.

In the following example the circulation is not operated by a 50-operation but by a non-prepared non-clearing multiplication.

n	)a(/41
n+1	)b(/40
n+2	)c(/65
n+3	)k(/10

In this example the following activities are performed in the registers:

C	B	A	R	
n/20				
)a(/41	n/20			
n+1/20		a	a	
)b(/40	n+1/20			
n+2/20	a	b		
)c(/65	n+2/20			
n+3/20	b	sgn b	a	
n+3/20	k	s	s	
)k(/10	n+3/20	k		k → A !!
n+4/20				k → store

The special feature of these instructions is that independent of the 50-operations they activate the circulation, which is necessary to obtain the head. In these non-prepared non-clearing multiplications a single-length number is added to a double-length product in the tail, carrying over into the head. This is the only occasion on which a double-length addition is effectuated. So all double-length additions must be programmed by means of the instructions for multiplication.

The instructions 54, 55, 56, and 57 shift a mutilated number from A to B. These mutilated numbers can nevertheless be used, if only it is exactly known in what manner the mutilation is effected. The rules holding in this case are very complicated. That is why the practical rule has been accepted that these operations may not be used for non-clearing multiplications. On further consideration it will be realised that by means of the operations 52, 54, and 56 no more can be obtained than by means of 50. Therefore they are never used.

1.44 Division

As in the division process the quotient originates in B, all the divisions require the prepare instruction to make the quotient go again from B to A. In a certain respect the division is the reverse of the multiplication. So in general a double-length dividend must be divided by a single-length divisor. The divisor is placed in R, the head of the dividend (shifted over one place to the right) is set up in A and the tail of the dividend is issued by the store during the division process. This tail has a separate sign digit, which, however, must not play a rôle in the division process. The sign of the tail is automatically ignored because the first digit taken into A is coming from A_{31} and not from the store. Normally it may be assumed that the sign of the tail is the same as the sign of the head of the multiplicand. The instructions 72 and 73 can serve to take in the tail of the dividend if its sign does not correspond with the sign of the head. A specific example of the dividing process is the following:

n	)k(/40
n+1	)d(/51
n+2	)s(/71
n+3	)q(/10

In this process the following activities are performed in the registers:

C	B	A	R
n/20			
)k(/40	n/20		
n+1/20		k	
)d(/51	n+1/20		
n+2/20	k	d	d
)s(/71	n+2/20	k	
n+3/20	q	r	r

C	B	A	R	
)q(/10	n+3/20	q	r	
n+4/20				q→store

It is possible to make use of a non-prepared division. In such a case it is impossible to obtain the quotient so that the instruction is only practical, if we want to know the remainder. As the division still considers the shifted contents of A to be the head of the dividend, this division is only of practical importance, if the head is zero and the dividend consequently is positive. For arithmetic problems, which require operations on numbers modulo n the non-prepared division has some sense.

1.45 The prepared jump

The third and very important application of the 50-operation is the use before a 20-operation occurring in the programme. In organising programmes we use sub-programmes for small parts of the problem, which must be constantly repeated (e.g. the determination of a sine). A programme (main programme) must have the facility of calling in such a sub-programme, which, after it has finished its actions, must return to the main programme and proceed at the location following the location where it came from. Often a datum is carried on to a sub-programme, from where it may return with one or more results.

The 50-operation provides a simple means to obtain the intermediate jump instruction, which is present in B, into A through the circulation. The standard equipment of the main programme and the sub-programme will always be:

n	)x(/51
n+1	s /20
n+2	etc.

s	s+m/10
⋮	⋮
(s+m	)

Then the following takes place:

C	B	A	R	
n/20		a		
)x(/51	n/20			
n+1/20	a	x	x	
s/20	n+1/20	a		
s+m/10	s/20	n+2/20		
s+1/20				n+2/20 → s+m
⋮	⋮	⋮	⋮	
s+m/20		F(x)	G(x)	
n+2/20	s+m+1/20			
etc.				

If as first instruction in a sub-programme a 20-operation should follow, the return instruction in A is maintained all the same, because after the 20-operation the circulation is switched off.

1.46 The round-off operations

In the multiplying process it is necessary to have the facility to round off products of double length to products of single length. This is not possible during the multiplying process because there is no input free on A_1 to add the round-off. After the multiplication the product is split up into two parts so that it is no longer easy to add a round-off to the tail with carry-over to the head. That is why a special round-off instruction has been provided.

The rules for rounding off in the inverse system are the following:

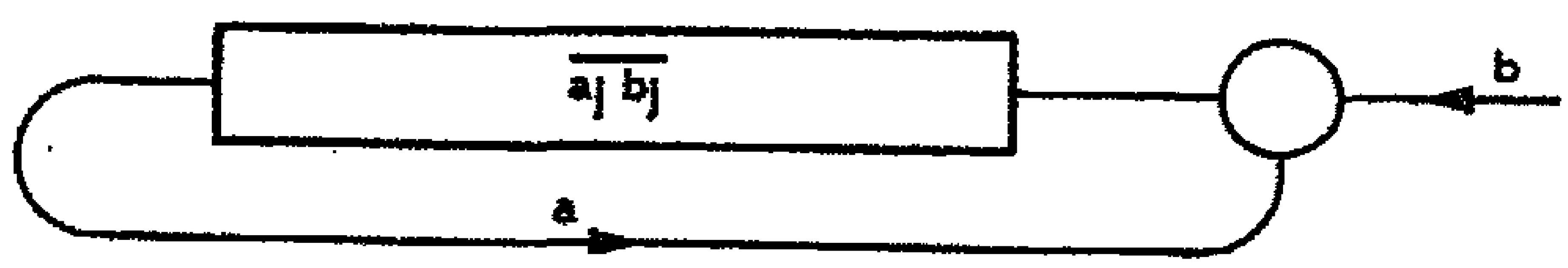
If $k \geq +0$: $k' = k + \varepsilon$ if $|s| \geq 2^{-1}$; otherwise $k' = k$

If $k \leq -0$: $k' = k - \varepsilon$ if $|s| \geq 2^{-1}$; otherwise $k' = k$

It is this very function that is performed by)ε(/25. As some instruction digits are used functionally, all the variants are present without extra equipment being required. With some variants special tricks can be done.

1.47 The conjunction operations

The operation conjunction performs the logical multiplication of two numbers digit by digit and is useful for the cutting away of parts of an instruction. The structure is as follows: The number already present in A opens or closes a gate which either or not passes the digits coming from the store.



In the table the result of a conjunction is represented by:

$$\text{conj } (n);(A) = \sum_{j=1}^{30}\{n_j a_j - n_o a_o\} = \overline{n_j a_j}$$

1.48 The stop operations

As to their arithmetic action the stop instructions are perfectly identical to the corresponding instructions 40, 42, 44, and 46. Moreover, they stop the machine. After the start key has been depressed, the machine resumes the programme. The stop instructions are useful to be written instead of a 40-instruction at critical places in order to give visual indication of important intermediate results when a programme is being tested.

1.49 The input operations

When data are being put into the machine, conversion from decimal to binary scale must normally be performed. This conversion can best be carried out in the arithmetic unit itself, but then the input instruction must preferably be adjusted to this conversion, which can be made most easily by multiplying the symbols which are read consecutively, directly during the reading process by the corresponding power of 10 and to add the result in the accumulator. Technically this can be realised very easily by letting the punched-tape reader replace the relay register. Then the multiplication is made quite analogously to the normal multiplication with the restriction that a double length product is not needed in this case. That is why the accumulator is only used over single length and the operation need not be performed in two phases.

The drawback of the before-mentioned system of conversion is that a fixed amount of digits has to be put in. Therefore it will be more convenient in practice to multiply a symbol by 1, to add it into A and to multiply the number obtained by 10. After this the next digit can be read in again. This process can best be performed cyclically. Then there must be a possibility to leave this cycle, and this possibility can be found in the fifth hole of the tape. For the representation of the digits in the decimal system only four holes are required. The presence of the fifth hole can be used as a marking. The input instruction tests on this marking, and according to it skips or does not skip the next instruction.

The symbols on the punched-tape are represented by:

$$u_4,u_3,u_2,u_1,u_o = u_4,\overline{u} = u_4, \sum_{j=0}^{3} u_j 2^j$$

From the table it can be seen that $\overline{u}.(n)$ is always added. Subtraction would be seldom of any use. All the reading instructions fill R, because it must be possible to multiply by 10 immediately afterwards. After having read the symbol the tape steps to the next symbol.

In the notation on paper the presence of the fifth hole is denoted by a stroke (/) behind the symbol. As the tape is hexadecimal it is sometimes useful to be able to denote the symbols $\geqslant 10$:

10 = a ; 11 = b ; 12 = c ; 13 = d ; 14 = e ; 15 = f

1.491 The output operations

At the output numbers must be deconverted. Just as it is possible to make the input instruction specially suitable for conversion, the output instruction can be made specially suitable for deconversion. The deconversion process is based on multiplying a fraction by 10. Then the head of the product is a deconverted binary coded digit and the tail must be set up for the next multiplication by 10. So the instruction is essentially a multiplication. Therefore the action of the output instructions has been made identical to the action of the corresponding multiplication instructions. The typewriter always types the tetrad formed by the rightmost 4 figures of the head. If the punch is switched on and punching is not suppressed, it punches the pentad formed by the rightmost five digits of the head.

If a number of typing instructions would be given in succession, only 75 ms would be available for each symbol, if the speed of the 11-instruction would be the same as the speed of the normal multiplication instruction. The typewriter can, however, not follow this speed. Therefore an extra waiting phase, which does not affect anything, has been inserted into the typing process between the first and the second phase.

The various tetrads which are possible are typed on the typewriter as follows:

0000 :	types 0	1000 :	types 8
0001 :	" 1	1001 :	" 9
0010 :	" 2	1010 :	" +
0011 :	" 3	1011 :	" -
0100 :	" 4	1100 :	" .
0101 :	" 5	1101 :	carriage return, line feed
0110 :	" 6	1110 :	tabulates
0111 :	" 7	1111 :	types space

When intermediate results are typed out, it is often desirable to type but not to punch the argument, the space between two numbers, extra line feeds in the lay-out, etc., so that the tapes produced thus can without further measures be used as input data. For this purpose a facility has been provided to suppress punching. If the intermediate jump instruction is a 20-instruction, typing and punching take place; if, however, the intermediate jump instruction is a 20-instruction + 2^{-12}, punching is suppressed. A suppression jump already affects an immediately preceding type instruction. This is caused by the fact that the typing actually takes place after the third phase during the subsequent jump. The same also holds for the releasing of the punch suppression with a normal 20-operation.

The 20-instruction + 2^{-12} is written in the programme as 28 and read in by the input programme as "octal digit greater than 7". It is possible that the punch suppression occurs together with the testing on A. Then a 24-instruction + 2^{-12} is required, which can be written as 2c.

1.492 The 29-operations

It is often desirable to have the facility of interfering in the course of the programme, dependent on facts observed visually (e.g. sub-tabellation in the neighbourhood of a maximum, suppression of undesired intermediate results, etc.).

For this purpose the 21-operations have been provided with a special device. When a 21-operation is increased by 2^{-12} (which may be written as 29) the functioning as regards the control will be:

$$n/29 : (B) + \varepsilon(1 + x_I) \rightarrow C$$

in which $x_I = 0$ or 1 dependent on the position of a switch which can be operated from the keyboard. Thus the course of the computation can be influenced during the operation of the machine. This switch is called selection switch I, and is functioning when c_{12} is present in the operation.

In the same manner the selection switches II and III have been provided on c_{11} and c_{10}. Their function is:

$$n + 2^{-11}/21 : (B) + \varepsilon(1 + x_{II}) \rightarrow C$$

$$n + 2^{-10}/21 : (B) + \varepsilon(1 + x_{III}) \rightarrow C$$

For the rest the operation functions as a normal 21-instruction.

1.5 The manual keyboard

PTERA is operated by means of a keyboard containing the following keys:

The start key. The function of this key is self-explanatory.

The stop key. By means of this key a programme can be stopped. If the stop key is being depressed while the machine has already been stopped, a single instruction is being carried out. This step by step operation is often useful for checking a programme or the machine.

The clear key. By means of this key in fact the register C is cleared by blocking off the entrance of C and starting the machine. So in C the instruction 0/00 comes, which, because (0) = 0 also clears A and B. Furthermore the control is brought into a condition as if the previous instruction has been a 20-operation. See further p. 31.

The three selection switches. The function of these switches has already been explained with the 29-operation.

The only function of the other keys on the keyboard is to provide the facility of loading the machine if the whole store is empty, and the facility of executing instructions manually to test the machine. A switch called OV deprives the machine of its autonomy. In that case a new instruction is not extracted from C but from a set of switches on which an instruction can be set up manually. There are eighteen of such switches, i.e. twelve for the address and six for the operation. A switch called AH provides the facility of switching over from the store to two keys "0" and "1" by means of which a word can be brought manually into the accumulator (or any other destination which is normally possible).

1.6 Programming

In the paragraphs which are now following, some examples of programming will be given. These examples mainly serve for the elucidation of the system of conventions as it has been adopted in the use of PTERA. It is by no means the purpose to give an exhaustive account, rather than to highlight some programming peculiarities of the PTERA code.

1.61 The input programme

So far only the functioning of the machine as such has been dealt with. The various instructions and their action have been discussed elaborately. So these are only the potential possibilities. An automatic computer owes its value mainly to the library of sub-programmes, which must, however, be written in a general form independent of the actual location where they are put into the store in a particular application. This form is determined by the system of conventions, which is not directly determined by the machine. It is possible to build up various systems of conventions, but for each type of machine it holds that the instructions are most suitable for a specific use. Or rather: the system of instructions has been designed beforehand in view of a particular system of conventions. Such a system is always a compromise between what is easiest for the programmer and what is simplest for the machine.

The structure of the system of conventions is mainly determined by:
a. the use of sub-programmes;
b. the input programme.

a. In this machine jumping to a sub-programme and storing the return instruction by the operations 50 and 20 is a very easy process. That is why all the independent parts of a computation will be brought in the form of a sub-programme. In R a datum can be taken along and in A and R two results can be carried back. If only one result need be carried back this will always take place in A and R. Series of coefficients, series of working registers, etc. will often also be brought in the form of a sub-programme, though execution of these programmes is out of the question. Then they can be used by means of the normal code digits and serial numbers.

b. The input programme is used to feed programmes and numbers into the machine. The most important thing is that the numbers and addresses must be converted from the decimal into the binary system. A second function of the input programme is to translate the general coding of all sub-programmes on the tape into the specific addresses in the problem concerned. All the tapes are coded beginning at 0 relative. While these relative addresses are being put in, they are adapted to the place where the programme will really be placed. This is given in a so called fixed parameter. There are eight of these fixed parameters (the reference points with respect to which the relative numbering takes place) available.

The requirements that we shall impose on the input programme are:

1. It must start on register 0, so that with cleared A, B and C the machine begins at 0 when the start key is being depressed.
2. It must be possible to put in the addresses of the instructions in such a manner that no non-significant zeros need be punched.
3. A code digit indicates with respect to what parameter the beginning must be counted. *)
4. By means of an input indication it must be possible to start input at an arbitrary location; such an input indication must also be able to use the parameters. Another type of input indication must start carrying out a programme that has been put in, at a certain address.
5. It must be possible to put in numbers by means of the normal input programme during input of instructions.
6. When special activities have to be performed during input, it must be possible to leave the input programme without destroying the possibility of continuing input at the point where it was interrupted.

The input programme runs as follows:

→	"0	0/00	Stop. Start of input programme.
	1	13/20	Jump to 13.
	"2	ε	
	"3	μ	$\mu = \varepsilon - 1$
	"4	2^{-18}	
	"5	2^{-15}	
	"6	2ε	Constants
	"7	$10^{9}\varepsilon$	
	"8	2^{-1}	
	"9	$10^{8}\varepsilon$	
	"10	10ε	
	"11	16ε	
	"12	35/44	
1 →	13	35/10	Clear 35
31, 68 →	14	2/03	Read code digit
	15	45/20	If stroked: jump to 45

*) D. J. Wheeler. Programme organization and initial orders for the EDSAC. Proc. Roy. Soc., A202(1950)573.

14 →	16	12/44	Form parameter instruction and put it in 22
72 →	17	22/10	
	18	20/20	Jump to 20
21 →	19	10/60	Read address
18 →	$\underline{20}$	2/0$\underline{5}$	
	21	19/20	
20 →	(22	35+c/44	Add parameter
	$\underline{23}$	4/0$\underline{7}$	Read left-hand operation digit
	24	61/20	If stroked: invert word
23 →	$\underline{25}$	5/0$\underline{7}$	Read right-hand operation digit
	26	30/20	If stroked: replace store instruction
25, 62 →	(27	n/10	Store instruction. (n/10 or n/20)
48 →	28	27/40	Add 1 to store instruction
	29	2/44	
26 →	30	27/10	
	31	14/20	Return to 14

Each instruction on the tape is written as:

code digit address / operation

The meaning of this is:

$$c\ a/op = (35 + c) + a/op$$

So when the address is being put in it is increased by the contents of register 35 + c, the register where the cth parameter is located. Parameter 0 in 35 has been reserved for absolute addresses, so 35 is always cleared. 36 has been reserved for the beginning of each sub-programme during input of that sub-programme.

The actual input programme starts at 14. The code digit c is read. If the code digit has no stroke, 15 is skipped. On 16 the instruction 35 + c/44 is formed, after which it is placed in 22. Then the address is read in and converted by means of the cycle 19, 20, and 21. The last digit has been marked off by means of a stroke at which it jumps from 20 to 22, where the so-called parameter instruction is carried out. The parameter is added, the first order digit is read into the machine in the right place, 24 is skipped, the second operation digit is read and 26 is skipped too. On 27 there is the so-called store instruction, which puts the instruction into the store. On 28, 29, and 30 the store instruction is augmented by 1, so that the next instruction is put into the next register. After this the procedure is repeated.

To determine at what location input must be started, an input indication is given at the beginning. By an input indication are meant all the tape combinations which ultimately do not enter the store. By stroking the last operation digit the storing instruction is replaced. So when n/10/ is on the tape, n/10/ is taken in, 26 is not skipped, and (30) replaces (27) by n/10, after which the input starts at n.

This is also the method to leave the input programme in order to carry out the programme at m. Then the storing instruction is replaced by m/20. Another instruction has to be taken in as yet, after which the programme runs into 27 and jumps to m.

When the machine is being started, the process is as follows: A, B, and C have been cleared altogether by means of the clear key and the machine has been brought into the condition as if the previous operation has been a 20-instruction. Then the following takes place:

C	B	A
0/00	0/00	0/00
1/00	0/00	0/00
0/00	0/00	13/20
0/00	13/20	0/00
13/20	0/00	0/00
35/10	13/20	0/00
14/20	etc.	

The first normal instruction is the clearing of 35, as is always required. After that the programme starts on 14.

In the starting procedure there is the exceptional situation that the "operation cycle" occurs some times in succession.

For various reasons it is desirable to be able to leave the input programme and to recontinue it without affecting the storing instruction. This is done by means of the stroked code digits. This is one of the reasons why the code digit is placed in front. Then already from the beginning it is known whether a normal instruction or a stroked code digit is following. The function of the stroked code digits is determined by the contents of the first track (32 - 63), running as follows:

32			
33		Available for special purposes	
34			Code digit
"35	0/00		0
(36		Begin address of sub-programme.	1
37			2
38			3
39			4
40		Parameters	5

	41			6
	42			7
	43			8
	"44	48/20	Constant	
15 →	45	44/44	Form and place jump instruction	
	46	47/10	48 + c/20	
	(47	48+c/20	Variable jump	
0/ →	48	28/20	Put next word into next register. Also available for special purposes.	
1/ →	49	69/20	Use directory	
2/ →	50	27/40	Remove store operation from instruction	
	51	4/46		
	52	36/14	Place this address in 36 and jump to directory programme	
	53	64/20		
6/ →	54		Available for special purposes	
57 →	55	10/60	Read digits of fraction	
8/ →	56	2/05		
	57	55/20		
56 →	58	7/51	Divide by conversion factor	
	59	61/70		
	60	0/05	Provide number with sign	
24 →	61	θ+2/63	$\theta + 2/63 = 5000031268 \approx \frac{1}{2}.10^9.2^{-30}$	
e/ 60 →	62	27/20	Jump to store instruction	
f/ →	63	0/20	Stop	

If with (14) the input programme takes up a stroked code digit c/, then the programme jumps to 45. There 48 + c/20 is formed and this jump is carried out.

The following code digits are provided:

0/: The utilization of this digit does not affect the register. Input is normally continued on the next register. Because it is rarely used, it is also available for other purposes.

1/: Make use of the directory } See next paragraph.
2/: Adjust the directory }

6/: Free for special purposes.

8/: Input of fractions with sign. The fractions are put in as natural numbers (of which the zeros on the left-hand side may be omitted). With (58) and (59) the division is made by the conversion factor $10^9.2^{-30}$, where (61) serves as tail of the dividend. On the free digit places 0 - 12 the instruction 2/63 has been supplemented by such digits that it is as much as possible equal to $\frac{1}{2}.10^9$. In this way the quotient is rounded off. (60) and (61) provide the number with sign, 0/ being equal to + and 0 being equal to -. Finally the fractions are put into the store by the normal store instruction.

Examples:
+0.376524276 is put in as 8/376524276/0/
-0.0005273 is put in as 8/527300/0

a/: b/: These two code digits can be released to take in fractions by means of a special input programme, a/ being equal to + and b/ being equal to -. Zeros on the right-hand side may be omitted; the last digit is stroked.
Examples:
+0.0003456 is put in as a/0003456/
-0.25 is put in as b/25/
With this coding, numbers punched out by the machine can immediately be read in in the same code (a = +, b = -). Moreover the programme is much quicker than the normal 8/ programme.

e/: Return immediately to the store instruction. By means of this code digit a register is cleared during input. After an input indication n/20/ e/ the machine directly starts executing n/20.

f/: Stop the machine.

Negative integers are put in by means of the jump on 24. By stroking the first operation digit of an instruction the address is put in as a negative integer. So -623 is put in as 0 623/0/.

1.62 The directory programme

A complete programme will as a rule consist of a number of sub-programmes and a main programme, which coordinates the functioning of the sub-programmes. In various cases standard sub-programmes from the library can be used; Some sub-programmes will have to be made specially for the problem. Sub-programmes which are not using other sub-programmes will be called sub-programmes of the 0^{th} order; sub-programmes which in their turn make use of a programme of the 0^{th} order, will be called programmes of the 1^{st} order, etc. The main programme is of the highest order.

In all the sub-programmes and in the main programme the storage locations are numbered from 0 (relative with respect to the beginning of this sub-programme). The initial address of a sub-programme will always be registered in 36 so that with the code digit 1 the relative addresses, when they are put in, will be adapted automatically to the location where the programme enters the store. These initial addresses need, however, not be known to the programmer. It is also more convenient if the length of a sub-programme need not be taken into account. The most logical method is to place all the sub-programmes consecutively into the store. Then each sub-programme can be given a serial number. The directory programme can automatically register the initial addresses of the consecutive sub-programmes. If in a sub-programme reference has to be made to a sub-programme of lower order, the latter can only be looked up in the directory if it has already been put into the store. Therefore the following convention has been adopted: sub-programmes of a lower order are always inserted before sub-programmes of a higher order. So the main programme always comes last.

The directory programme is controlled by means of the code digits 1/ and 2/. The code digit 2/ provides for the registration in the directory programme; by means of 1/ the directory is made use of. The code digit 2/ has the following functions:

a. At the end of each sub-programme the next location is recorded in 36 as the beginning of the next sub-programme;
b. the address is also recorded sequentially in the directory.

It is more correct to give the 2/ at the end of a sub-programme, because there can be intermediate activities such as the insertion of parameters. In that case the store instruction is lost, but the correct address is retained in 36. (The code combination GK in the EDSAC, which is more or less comparable, is given at the beginning of a sub-programme.) Only the main programme does not terminate with 2/.

A code digit 1/ always forms the beginning of a whole instruction. Such an instruction is composed as follows:

1/serial number/address/operation

in which the serial number may run from 1 to f. (0 can be used for special applications.) The meaning is:

$$1/r/a/op = (74 + r) + a/op$$

Then the address, relative with respect to the beginning of the said sub-programme, is put in. If the last operation is stroked, a 1/-instruction can also be an input indication.

The directory programme runs as follows:

53 → (64	75/10	Place address into the directory
65	2/40	Advance the directory store instruction by 1
66	64/44	
67	64/10	
68	14/20	Resume input programme
1/ → 69	2/01	Read serial number (one digit only)
70	4095/20	If digit not stroked: jump to the last register
69 → 71	73/44	Form look-up instruction and jump to 17
72	17/20	
"73	74/44	Constant
74		
75 ⋮ 90		Directory

The action is as follows: With 2/ the programme enters at 50. The storing instruction, which is in 27, and which has already been increased by 1, is read in and the operation part is removed from it. Starting from 75 it is put into the directory and stored at 36. (65) to (67) provide for the advancing of the directory store instruction.

With 1/ the programme arrives via (49) at 69 where the serial number is read. The latter is normally stroked. At 71 the instruction 74 + r/44 is formed, which is stored by (17) in the location of the parameter instruction. For the rest the input of an address and an operation proceeds normally.

On 70 a spare facility of the directory programme has been utilized to jump with a non-stroked serial number, to the last register of the store. If there is a 20-instruction in the said register, it can be used for special purposes. The instruction 4095/20 provides, however, also the facility to interrupt the normal alternation: instruction cycle — operation cycle in another way than a jump. By placing an instruction 0 0/40 in 1023 the following takes place: The jump 4095/20 functions as 1023/20, because the digits 18 and 19 of the address do not exist. A is cleared by the instruction 0 0/40. The intermediate jump operation 4095/20 is increased by unity resulting in the instruction 0 0/30. So the intermediate 20-operation has been cancelled. 0 0/30 puts 0 into A and also shifts a 0 to B. Thus C is also cleared and the machine is stopped, with A, B, and C cleared.

1.63 The pre-input programme

The input programme must somehow have come into the store, e.g. by putting it in manually, which entails much work. It is much more convenient to make use of a pre-input programme, a very simple input programme, which by means of a specially prepared tape is able to take in the normal input programme. The pre-input programme itself must be as short as possible and the tape may not be too long. This appeared to be realizable only by applying a cascaded pre-input programme. The first pre-input programme first takes in another input programme which works a little faster and with an easier code. The second pre-input programme, which has thus been formed, is extended a little and then it takes up the normal input programme in a special code.

The second pre-input programme runs:

"11	16£	Constants
"32	£	
35 → 33	11/60	Reading cycle. Read all the digits but one in the hexadecimal system
39 → 34	11/05	
35	33/20	
34 → 36	32/07	Read last digit
37	38/10	If the digit is stroked: replace
36 → (38	n/10	store instruction
39	34/20	Return to reading cycle

In the first place a new input indication must always be given for the storing of an instruction or number. The first activity of the programme is to extend itself by:

40	32/44	Augment store instruction by 1
41	38/10	
42	34/20	Return to reading cycle

Then (37) is replaced by:

37	41/20	Jump to 41

and (39) by:

39	38/40	Take in the store instruction

With this last addition the pre-input programme has got the facility of advancing its store instruction.

The first pre-input programme is:

"0	1/07	Read digit times (1)
"1	0/10	Constant = 2^{-18}
2	0/07	Read digit times (0). Test on stroke
3	4/10	Store the store instruction

In principle it is already possible to take in a tape by means of this programme. When the machine starts on 0 with 95/ on the tape: (4) = 5/14, after which the machine stops on 5. *) 5 becomes cleared. After that the machine is started for the second time on 0 with 20 on tape. Then (5) = 0/20, and the programme can return itself. 0/20 must be brought in C if the machine is to start on 0. It does not take more work to put this 0/20 directly in 5. Then the machine need be started only once. So the following program is inserted manually:

"0	1/07	Read digit times (1).
"1	0/10	Constant
2	0/07	Read digit times (0). Test on stroke
3	4/10	Replace store instruction
(4		Store instruction
5	0/20	Return to 0

The tape reads:

4	1c/	= 12/14		Thus (A) will be cleared
4	ebebdb/	= 33/14	(A) = 0	12 is a rubbish dump
				Now the new store instr.

*) The idea to place a 14-instruction in 4 is from Dr G. v. d. Mey.

			can be built up
33	fbf0	= 11/60 =(A)	Instr. to be put in, from which exclusively 16/14 can be built up
4	35/	= 16/14 (A) = 0	
4	8b8b1c/	= 34/10 (A) = 0	
34	0b	= 11/05 (A) = 0	
4	1c/	= 12/14 (A) = 0	
4	0c0c1c/	= 36/14 (A) = 0	
36	fffff2b0	= 32/07 =(A)	
4	13/	= 35/14 (A) = 0	
35	ab0b0b	= 33/20 =(A)	
4	75/	= 38/14 (A) = 0	
38	cbd0/	= 11/10 =(A)	
4	00/	= 11/10 (A) = 0	
11	40	= 0/40 (A) = 0	
4	1c/	= 12/14 (A) = 0	
4	3c3c3d/	= 37/14 (A) = 0	
37	ccccced0	= 38/10 =(A)	
4	04/	= 42/14 (A) = b	
4	ddcd0d/	= 39/14 (A) = 0	
39	6b6b6c	= 34/20 =(A)	
4	00/	= 34/20 (A) = 0	
11	20/0	= $2^{29}\varepsilon$	The constant 16ε can be formed only in 3 steps, one constant always being needed for the input of the next one.
11	8/0	= 2ε	
11	8/0	= 16ε	
	102/0/	(38) = 32/10	
32	1000000/0	= $2^{28}\varepsilon$	ε too can be formed in 2 steps only
32	0/8	= ε	
	102/8/	(38) = 40/10	
40	2402/0	= 32/44	First the second input programme is supplemented by an advancing part
	102/9/		
41	102/6	= 38/10	
	102/a/		
42	202/2	= 34/20	
	102/5/		
37	202/9	= 41/20	
	102/7/		
39	402/6	= 38/40	

	0/0/	Now the input of the normal input pro-
	100/0/	gramme follows
0	0/0	
1	200/d	
2	0/1	
	1100/3/	Put in (3) negatively
3	3ffffff/f	The only way of putting in μ !
	100/4/	
4	100/0	
5	800/0	
6	0/2	
7	3b9aca0/0	
8	2000000/0	
9	5f5e10/0	
10	0/a	
11	1/0	
12	2402/3	
13	102/3	
14	1800/2	
15	202/d	
16	2400/c	Cf. pages 29 and 30
17	101/b	
18	201/4	
19	600/a	
20	2800/2	
21	201/3	
22	2400/0	
23	3800/4	
24	203/d	
25	3800/5	
26	201/e	
27	101/b	
28	401/b	
29	2400/2	
30	101/b	
31	200/e	
	200/0/	Stop
	0/0	

In the first part of this programme often an extra replacement of the store instruction is required to clear (A). The most important feature of this programme is that the building-up cycle is the same as the storing cycle. The building-up process can only be performed thanks to the 14-instructions in 4; clearing can only be carried out thanks to the 10-instruction in 3.

1.64 Some special input programmes

It has proved to be very useful to apply the serial input programme. It often occurs that large parts of programmes consist of series of instructions which resemble one another very much and which have a regular structure. This will especially be the case with stretched programmes. The serial input programme makes use of the free code digit 6/. The structure of such an instruction is:

6/ c a/op x/y/z/

In this instruction c a/op is a normal instruction with code digit, address and operation. This may also be a 1/ or even an e/ instruction. x is the number of times that the instruction must be inserted; y is the size of the interval between successive instructions of the same type and z is the increment by which the instruction must be increased every time. After the serial input the normal input is resumed consecutively. The programme itself does not enter the directory. The programme runs and functions as follows:

		0	66/20/	Do not enter this programme in the directory by not advancing the directory store instruction
		8/	1/0	
		0	54/10/	
6/ →	54	1	0/20	Go with 6/ to this programme
		1	0/10/	
54 →	0	0	27/40	Take store instruction and put it into 1.19
	1	1	19/10	
	2	0	0/50	Take in the instruction
	3	0	30/20	
0.27 →	4	0	27/12	Put it negatively into 0.27
	5	0	0/50	Take in x
	6	0	17/20	
0.22 →	7	0	47/10	Place x in 0.47
	8	0	0/50	Take in y
	9	0	17/20	
0.22 →	10	0	34/10	Put y into 0.34
	11	0	47/60	Form xy
	12	1	19/44	Form the last store instruction for test. Place this into 0.47
	13	0	47/10	

	14	0	0/50	Take in z
	15	0	17/20	
0.22 →	16	0	22/12	Place -z into 0.22
25 →	17	0	27/46	Advance and place instruction
	18	0	27/16	
	(19	e/		Store instruction
	20	1	19/40	
	21	0	34/44	Increase store instruction
	22	1	19/14	
	23	0	47/47	
	24	0	22/32	Test and take in z
	25	1	17/20	If not ready: return to 17
24 →	26	1	19/40	
	27	0	34/46	Place the normal store instruction
	28	0	29/20	again and return
		2/		

1.65 Programmes for floating addressing

When a programme is being made it often happens that some instructions have been omitted. It is very difficult to insert these instructions afterwards, as then the entire programme would have to be renumbered. WILKES *) has given some methods through which these difficulties can be overcome. It is, however, a drawback of these methods that they require a rather extensive administration programme and that a programme that has already been put in, must be corrected afterwards by this administration programme. For PTERA a somewhat simpler method has been developed.

The procedure of floating addressing is as follows: No addresses are punched on the tape. The instructions enter the store consecutively without further provisions having to be made. Labels need be given only to working registers, constants, variable instructions and points to which jumps are referring. There is a difference between references to registers which have already been put in and registers which have still to be put in. For the first kind the location to which reference has to be made, is already known in the machine. If an arbitrary number is allotted to such a register, further on reference can be made to that register by means of this number.

If, when making a programme, one must refer in the normal manner to places that have still to be filled in, the procedure

*) M. V. Wilkes. The use of a "floating address" system for orders in an automatic digital computer. Proc. Camb. Phil. Soc. 49(1953) Part 1, 84.

is mostly that the relevant instruction is temporarily not inserted, and is put in as soon as the point to which reference has to be made, is known. This method can also be applied to floating addressing. The advantages of such floating programmes are:

1. Instructions that have been forgotten, can be inserted without further provisions. This does not affect the coding on the tape of the remaining part of the programme.
2. The floating addresses which are allotted, are perfectly arbitrary numbers. A certain label can be used over again when it is no longer needed for reference to the former location. Thus the labels often consist of only one digit, which is favourable for the input time.

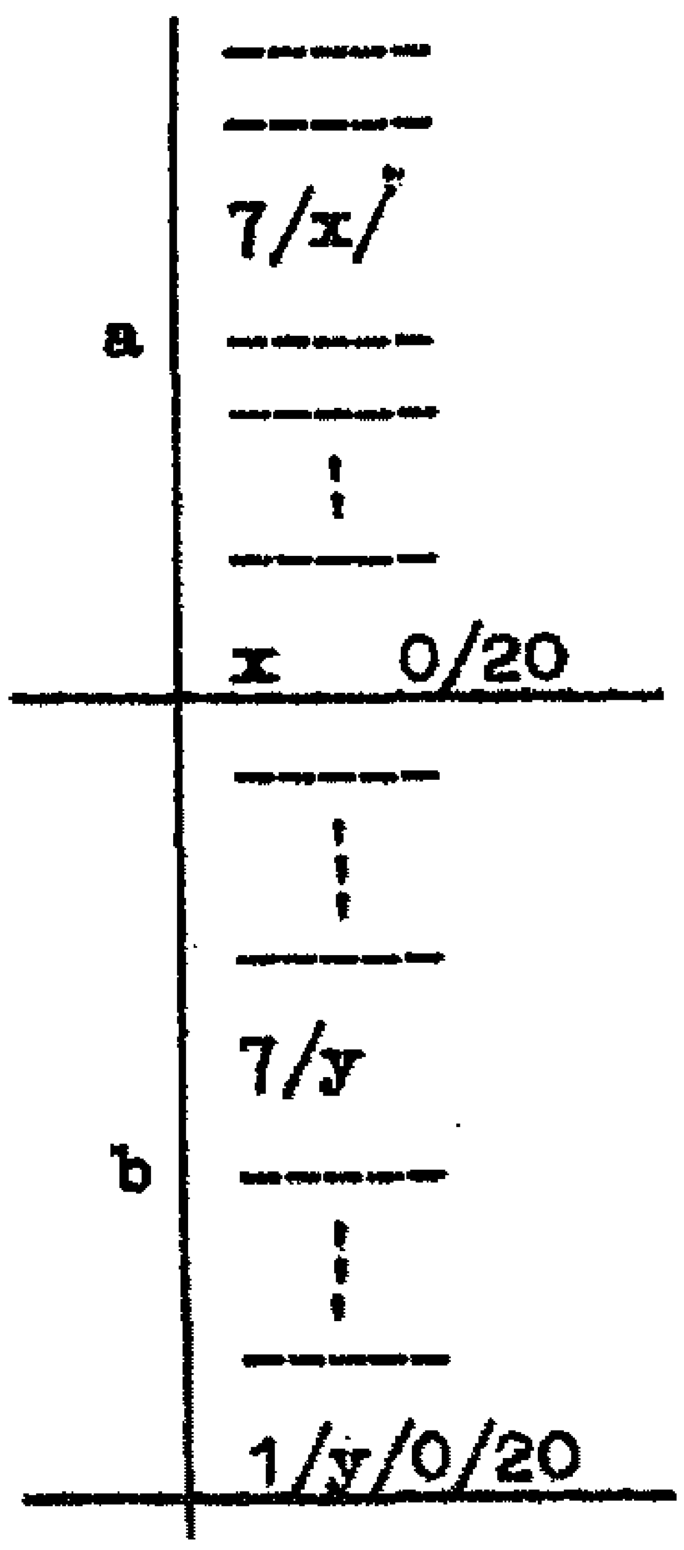
The programme that provides for the floating addressing, makes use of the code digits 0/ and 7/.

The 7/ part is used to allot floating addresses to certain registers. For this purpose use is made of the parameter registers and of the directory registers. If we introduce the abbreviation: $\langle x\rangle$ = the address part of (x), then the operation of the 7/ combinations can be expressed as follows:

$7/x/ \;:\; \langle 27\rangle \rightarrow 35 + x \qquad\qquad x = 1(1)8$

$7/y \;:\; \langle 27\rangle \rightarrow 74 + y \qquad\qquad y = 0(1)f$

These combinations are applied as follows:



By means of this combination the real address a is recorded as floating address x

By means of this combination a jump is made to the real location a.

By means of this combination b is recorded in 74 + y in the directory

By means of this combination the recorded address is used again

So with these combinations alone, it is only possible to use addresses that have already been filled in.

Reference to addresses which have yet to be inserted, is made by means of the 0/ combinations. The function is:

$0/x/op \;:\; \langle 27\rangle/op \rightarrow (35 + x)$

$0/y\ op \;:\; \langle 27\rangle/op \rightarrow (74 + y)$

These combinations are used in the following manner:

	7/x/	The location of the instruction referring to a location that has yet to be filled in, is kept free. The real location is a. The location to be referred to is in reality n
a	e/	

	⋮	
n-1	-----	
	0/x/op	Before n is going to be filled in, n/op is filled in on a by means of this combination.
n	-----	

The programme which provides for the code digits 0/ and 7/ runs:

	0	66/20/	Do not enter programme in directory
	0	1/0/	
	0	48/10/	Place 0/ entrance
0/→0.48	1	9/20	
	0	55/10/	Place 7/ entrance
7/→0.55	1	0/20	
8/→ 56	0	0/50	Replace 8/ part
57	0	17/20	
	1	0/10/	
0.55→1.0	0	2/01	Take in x
1	1	8/44	If not stroked: form 74 + x/10
0 → 2	0	d/44	If stroked : form 35 + x/10
3	1	6/10	Place this in 1.6
18 → 4	0	27/44	Form ⟨27⟩ or ⟨27⟩/op
5	0	4/46	
(6	e/		Store this with 35+x/10 or 74+x/10 or (35+x)/10 or (74+x)/10
7	0	e/20	Return to input programme
"8	0	39/00	Constant
0.48 → 9	0	2/01	Take in x
10	1	8/44	If not stroked: form 74 + x/44
9 → 11	0	c/44	If stroked : form 35 + x/44
12	1	d/10	Place this in 13
(13	e/		Extract (35 + x) or (74 + x)
14	0	4/44	Form and place (35 + x)/10 or (74 + x)/10
15	1	6/10	
16	0	4/01	Take in operation digits
17	0	5/05	

1.18	1	4/20	Use 4 - 7 in common with first part.
	2/		

1.66 Rapid input programme for fractions

For programmes in which many fractions have to be put in the 8/ code is not the most suitable one. The greatest difficulty is that the zeros have to be supplemented on the right hand side of the number. It would be better if they could be deleted but then they have to be placed on the left hand side.

The notation of the programme meeting this difficulty is the following:
+ is code digit a/ , - is code digit b/ , then follow the digits of which the last one is stroked. So:

+0.3 becomes a/3/
-0.0095 becomes b/0095/

The programme runs:

	0	66/20/	Do not enter programme in directory
	0	1/0/	
	0	56/10/	
8/→0.56	1	0/20	Place 8/ entrance
	0	58/10/	
a/→0.58	1	6/20	Place a/ entrance
59	1	5/20	Place b/ entrance
	1	0/10/	
0.56→1.0	0	0/50	
1	0	17/20	
0.22→"2	0	7/51	Substitute for 8/ part
3	0	61/70	
4	0	60/20	
0.59 → 5	1	36/40	Form 0 7/51 for + and
0.58 → 6	1	2/44	form 0 7/53 for -
7	1	26/10	Place this into 26
8	0	9/03	
9	1	26/20	
8 → 10	1	30/07	
11	1	26/20	
10 → 12	1	31/07	
13	1	26/20	
12 → 14	1	32/07	
15	1	26/20	

14→1.16	1 33/07	Read digits
17	1 26/20	If stroked: jump to 26
16 → 18	1 34/07	
19	1 26/20	
18 → 20	1 35/07	
21	1 26/20	
20 → 22	0 10/07	
23	1 26/07	
22 → 24	0 2/05	
25	0 0/20	If last digit not stroked: stop
(26	e/	0 7/51 or 0 7/53
27	0 61/70	Divide by conversion factor
28	0 0/45	Take over into R
29	0 27/20	Return to store instruction
"30	0 10000000/00	
"31	0 1000000/00	
"32	0 100000/00	
"33	0 10000/00	Constants
"34	0 1000/00	
"35	0 100/00	
"36	0 0/02	
	2/	

1.67 Rapid input programmes for instructions

The normal input programme offers many facilities for the programmer, which, however, slows down the speed. For programmes which are frequently used and which consequently have often to be put into the machine, it is recommendable to have the machine punch out the programme in absolute form such as it is present in the store in such a code, that this tape can be read in as quickly as possible by means of a special input programme. As to speed, the input programme is most important here; the tape punching programme need be used only once. As only 4 out of the 5 holes in the tape are used numerically, it is evident that the hexadecimal system should be used. Only instructions (and all that can be written in that form) and numbers are distinguished. The rapid input programme is written over the directory programme, which has no function any longer. By means of 6/ it is called into action.

	0 73/10/	
83 → (72		Contains store instruction
73	0 72/40	

	74	0 2/44	Increase store instruction by 1
54 → 84 →	75	0 72/10	
	76	0 88/03	Read 1st digit times 2^{16}
	77	0 64/20	If stroked: read number
76 →	78	0 4/07	Read 2nd digit times 2^{12}
	79	0 14/20	If stroked: go to normal input programme
78 →	80	0 89/05	Read 3rd digit times 2^{8}
	81	0 11/05	Read 4th digit times 2^{4}
	82	0 2/05	Read 5th digit times 2^{0}
	83	0 72/20	If not stroked: go to store instruction
82 →	84	0 75/20	If stroked: replace store instruction
	"85	0268435456/00	$= 2^{28}$
	"86	0 16777216/00	$= 2^{24}$
	"87	0 1048576/00	$= 2^{20}$ Constants
	"88	0 0/02	$= 2^{16}$
	"89	0 256/00	$= 2^{8}$
		0 75/20/	Change over to high speed input
		0 49/10	

	49	02000	= 0 0/20	Put 1/ out of operation
		01034/	= 0 52/10/	
	52	01024	= 0 36/10	Change part for 2/
	53	0200e	= 0 14/20	
6/ →	54	0204b	= 0 75/20	Place entrance for 6/
		01040/	= 0 64/10/	
77 →	64	18055	= 0 85/03	1st digit times 2^{28}
	65	28056	= 0 86/05	2nd digit times 2^{24}
	66	28057	= 0 87/05	3rd digit times 2^{20}
	67	28058	= 0 88/05	4th digit times 2^{16}
	68	28004	= 0 4/05	5th digit times 2^{12}
	69	28059	= 0 89/05	6th digit times 2^{8}
	70	2800b	= 0 11/05	7th digit times 2^{4}
	71	28002	= 0 2/05	8th digit times 2^{0}
		00/	Proceed to normal input programme	

The absolute punching programme associated with the rapid input programme has been made as quick as possible with respect to the input programme. The procedure is as follows: after a programme has been inserted into the machine (36 is not affected), the programme asks for an initial address. From there the contents of the store are punched out. When the selection switch I is operated, the programme stops with the extraction instruction in A. After it has been checked visually whether a sufficient number of data have been punched, the process can be continued by switching off I. This can be done in two ways: with the selection switch II on 0 the process is continued from the place where the programme had stopped; when II is switched on, the programme asks again for a new initial address. This is useful to punch out a programme which consists of various non-sequential parts. When sufficient data have been punched, I is not switched off after which the machine is started. The programme fills in 36 and writes the input indication for starting on the tape. Furthermore blank tape is punched till I is switched off again. The machine stops on 0.

Pre-punching

	0 37/10/	
0.37	p/00	Location of the programme

Programme

	0 38/10/	
0.38	2 63/00	Parameters
39	0 0/8f	
	2 0/10/	
"2.0	0 20/00	Stop. Also constant
"1	2 1071 906 817/41	=2 1/41 + constant for punching 0/00/6/
2	0 11/13	Punch 0/
3	0 2/13	0
4	0 11/13	0/
5	0 11/11	6/
62 → 6	0 0/50	Take in initial point x
7	0 30/20	
8	0 4/45	Form x/10 in R
9	3 0/44	Form x/41 in A
37 → 10	2 18/10	Place extraction instruction. Also place return instruction
11	3 2/11	
12	0 11/11	
13	0 11/11	
14	0 11/11	Punch input indication
15	0 11/40	
16	0 0/50	
17	0 11/15	
(18	e/	Extraction instruction or return instr.
19	3 1/36	Test whether number is negative. Subtract 2^{20} (Test on A)

	2.20	2	50/20	If negative: punch as a number
19 →	21	0	11/30	If positive: test whether $\{(x) - 2^{20}\} \geqslant 0$ (Test on A)
	22	2	52/20	If positive: go with 16 to punching part for numbers
21 →	23	3	2/11	R has still been preserved
	24	0	11/11	
	25	0	11/11	Punch instruction
	26	0	11/11	
	27	0	11/11	
	28	2	18/40	
	29	0	2/44	Advance the extraction instruction by 1
	30	2	18/10	
	31	0	0/29	Test on selection switch I
	32	2	18/24	If 0: proceed
31 →	33	2	18/00	If 1: stop with extraction instr. in A
	34	0	0/29	Test on selection switch I
	35	2	60/20	If 0: jump to 60
34 →	36	3	3/51	Punch 0 36/10/
	37	2	10/20	
18 →	38	0	36/41	
	39	3	2/11	
	40	0	11/11	Punch (36)
	41	0	11/11	
	42	0	11/11	
	43	0	11/11	
	44	0	0/11	Punch 0
	45	3	4/41	
49 →	46	3	5/11	Punch 0/1 0/20/e/ and then blank until
	47	0	0/2d	the selection switch is back on 0
	48	0	0/20	Stop on 0
47 →	49	2	46/20	
20 →	50	3	6/47	Form $\{(x) - 2^{20}\} - \{\mu - 2^{20}\}$ by which the sign digit is removed from the number
	51	2	0/40	
22 →	52	0	0/50	Punch 0/ for pos. Punch 4/ for neg.
	53	0	2/15	The first symbol is always stroked
	54	0	0/50	Punch two binary digits behind the point
	55	3	7/15	So the second symbol also has a stroke
	56	0	11/11	0/ for pos., 4/ for neg.
	57	0	11/11	Punch 3 digits of number
	58	0	11/11	

2.59	2	24/20	Punch the remaining digits like those of an instruction
35 → 60	4	0/21	Test on selection switch II
61	2	18/24	If 0: return to extraction instruction
60 → 62	2	6/24	If 1: take in a new initial point
	3	0/10/	
"3.0	0	0/31	
"1	0	1048576/00	$= 2^{20}$
"2	0	0/40	
"3	0	36/10	
"4	0	538446366/00	Punching constant for 0/10/20/e/
"5	0	32/00	
"6	0	1072693248/00	$= \mu - 2^{20}$
"7	0	4/00	
	2	0/24/	
	e/		

Part 2. ZEBRA

Introduction

When a computer is being constructed it is not possible to vary the original projects very much. Still development is needed.

The principles on which PTERA is based, have given rise to the designing of two machines: machine ZERO, which has already been described by the author earlier *) and ZEBRA, which will be described in this part.

ZERO was designed as experimental machine for PTERA, and ed operated as such. The idea of using functional digits based operation was fully applied in it. The machine did not have a multiplier nor a divider. The programme required for multiplication was very long, so that the design was not suitable for a practical machine. In part 3 the machine ZERO will be used further as a starting point of some theoretical observations.

The point was whether with a consequent application of the practical use of the operation digits it would be possible to construct a practical and yet simple machine (e.g. without multiplier and divider), such with the elimination of some flaws inherent in PTERA, i.e.:

It is very uneconomical to use a whole revolution for the extraction of a number, while a circumference contains 32 numbers.

The parallel adding unit requires a great number of valves, which are more components that can go faulty than in a serial accumulator. In PTERA it has proved to be very difficult to reduce the troubles in the arithmetic unit.

Double-length addition can only be performed via the multiplication.

The A-accumulator is not used while the instruction is being extracted.

It has been tried to make a design in which the principles program control of PTERA, the functional use of the operation digits and the simplicity have been applied as much as possible.

The uneconomical use of time is mainly due to the way in which the numbers have been distributed over the circumference. If the numbers are not interspersed, the first difficulty can be mitigated for the greater part. To derive maximum profit from the L store with a relatively long access time, optimum processing (minimum latency coding) is required. For this purpose a number of immediate access registers must be available. reducing of the waiting time can be eliminated. **)

v. d. Poel. A simple electronic computer. Appl. Sci. (1952)367.

L. Freedman. Elimination of waiting time in automatic rs with delay-type stores. Proc. Camb. Phil. Soc.)426.

The second drawback can be avoided easily by having a delay-type accumulator with a serial adder. This fits quite well in the system of having the accumulator built in the same physical form as the immediate access registers. For the sake of simplicity multiplication should also be carried out in series, which in principle requires about as many word periods as there are digits in a number. As the word periods have, however, become a factor 32 shorter, multiplication need not be slower than in PTERA. (The preparation cycle and the intermediate jump may not be counted, because they are slow in PTERA but quick in ZEBRA.)

For a serial adder the complement system is more suitable, because the pseudo-complement system requires a second passage through the adding mechanism to account for the end-around carry.

The third problem can be surmounted easily by coupling the two accumulators, which are at any rate required to carry out a multiplication, in such a way, that a carry originating in the tail accumulator is added to the head accumulator.

The fourth drawback can be partly eliminated by the consistent use of functional digits also on the intermediate jumps. If the addresses for the two kinds of stores are also separated, even the instruction and the operation cycle can coincide, if one address is used for the source of the new instruction and the other address for the operand.

The ZEBRA is an automatic electronic computer, based on the following principles:

1. The operational part (control and arithmetic unit) is small with respect to the store, which has a homogeneous structure.
2. The structure of the operational part is most simple. No built-in multiplier or divider. Serial operation.
3. The number of elements has been reduced to a minimum, in order to obtain a maximum reliability.

As the structure of the machine is so simple the possibility of making mistakes is small and possible mistakes are easily located. With a very small part of the store plus a well operating operational part, the machine can locate its own troubles in the remainder of the store. Though no multiplying and dividing facilities have been provided, the multiplying programme need not be elaborate. Because of the enormous flexibility in the coding this multiplying programme can be very simple. Other facilities as normalizing, automatic modifying of instructions, repeating of instructions etc. can be solved very elegantly.

The possibility of optimum programming (minimum latency coding) yields a high speed of operation. The multiplying speed is less high. Special attention has been paid to increasing the speed of interpretive programmes for floating point operations and operations with double-length numbers.

A large class of programmes for ZEBRA is dead, i.e. nothing has to be written on the drum. *) This is largely because all action takes place in the short registers. Also the facility for modifying instructions is very helpful in this respect. The advantage of dead programmes is that the tracks on which they are present, can be switched off for writing, so that these programmes can remain permanently in the store. This applies for example to the input programme, test programmes, and standard output programmes.

The structure of an instruction is remarkable for its functional digits in the operation. The whole programming is in fact a micro-programming. **) The operations are analysed into much more elementary actions. This accounts mainly for the great flexibility of the code.

2.2 The structure of the machine

Numbers inside the machine consist of 33 bits, numbered from 0 to 32. A number p has the value:

$$p = \overline{p_0 p_1 \ldots\ldots p_{32}} = - p_0 + \sum_{j=1}^{32} p_j 2^{-j}$$

in words: for the representation of negative numbers the complement system has been used.

2.21 The store

The store is formed by a magnetic drum with a storage capacity of $8192 = 2^{13}$ numbers each of 33 bits. The speed of rotation is 6000 rpm; this is 10 ms per revolution. The number of tracks is 256 and the number of words per track is 32. Consecutive digits of a word are placed consecutively on the drum and consecutively numbered registers are also placed in a normal ascending sequence. The least significant digit of a word is coming first in time. A word time is $1/32 \times 10$ ms = 312 μs.

2.22 The operational part

The operational part consists of the following registers:

A: Register of 33 bits for the accumulation of the most significant part (head). (One extra bit for overflow. Cf. 2.41, R-digit.)

B: Register of 33 bits for the accumulation of the least significant part (tail).

C: Control register. This register receives the next instruction to be executed.

D: Auxiliary register for the control.

A and B are called accumulators.

---

*) W. L. v. d. Poel. Dead programmes for a magnetic drum automatic computer. Appl. Sci. Res., B3(1953)190.

**) M. V. Wilkes & J. B. Stringer. Micro-programming and the design of the control circuits in an electronic digital computer. Proc. Camb. Phil. Soc., 49(1953)230.

Short registers: Registers of 33 bits, numbered from 0 to 31 (to distinguish between drum addresses and short addresses, drum addresses will be written with 3 digits, or will be $\geqslant 32$). Only the short registers 4 to 15 have been materialized. All short registers are immediately accessible. They can be realised by means of delay lines on the drum *) or by magnetostrictive delay lines **).

Special short registers:

0:	$(0) = 0$	In these registers no writing can take place.
1:	$(1) = \varepsilon = 2^{-32}$	
2:	$= A$. The A-accumulator has the address 2.	
3:	$= B$. The B-accumulator has the address 3.	

4: Cf. the action of the A-operations.

5 to 14: normal short registers.

15: Cf. the action of the LR and XD-operations.

16 to 22: Not built in the machine.

23: $(23) = -1$ = the most significant one in the word.

24: (24) = conj $(A);(B)$. 24 always contains the logical product of (A) and (B). In 24 nothing can be written.

25: Set teletype signal equal to a_0. $(25) = (A)$ on reading.

26 to 31: Input and Output. Cf. 2.81 and 2.82.

2.3 The structure of the instructions

The digits of an instruction contained in C are called c_0 to c_{32}. The names and short characteristics are:

		0	1
c_0	A-digit	jump	add
c_1	K-digit	short register for accumulator	short register for control
c_2	Q-digit		$(B) \pm \varepsilon \rightarrow B$
c_3	L-digit		shift AB left
c_4	R-digit		shift AB right
c_5	I-digit	additive	subtractive
c_6	B-digit	use A	use B
c_7	C-digit	do not clear	clear
c_8	D-digit	read	write on drum

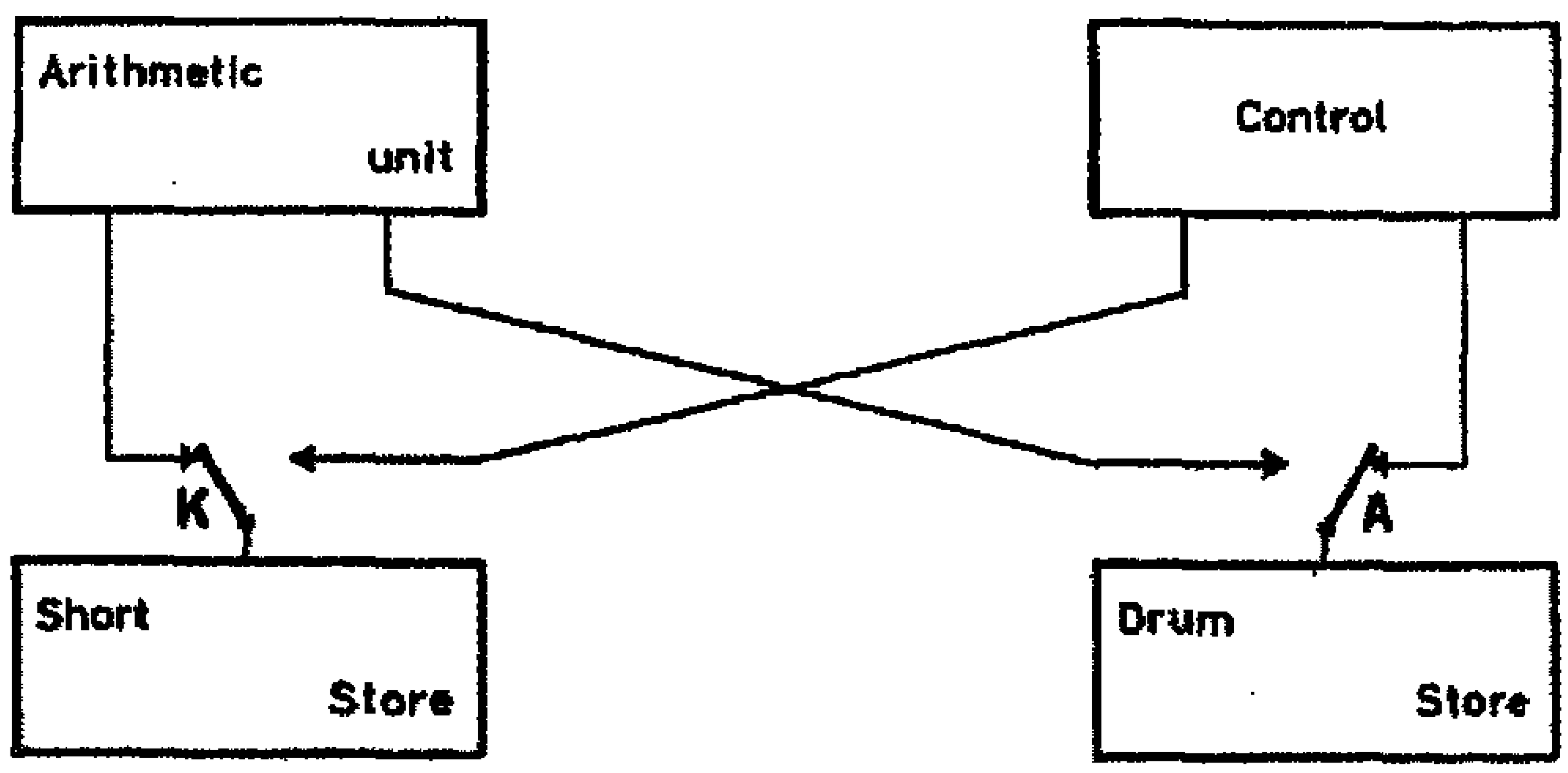
*) Staff of the Computation Laboratory. Description of a magnetic drum calculator. The Annals of the Computation Laboratory of Harvard University, Vol. 25. Harvard Univ. Press. Cambridge (Mass.) 1952.

**) E. M. Bradburd. Magnetostrictive delay lines. Electr. Comm. 28(1951)46.

c_9	E-digit	read	write in short store
c_{10}	V-digit	selection	test
c_{11}	V_4-digit		V4: test on b_{32}
c_{12}	V_2-digit	Un: test KSn	V2: test on b_o
c_{13}	V_1-digit		V1: test on a_o
c_{14}	W-digit	wait	do not wait for drum
$c_{15}-c_{19}$	Short address. Number of digits <3.		
$c_{20}-c_{32}$	Drum address. Number of digits $\geqslant 3$.		

The A-digit determines the character of the operation. If $c_0 = 0$ the operation is called X, and if $c_o = 1$ the operation is called A. An X-operation has as main element the extraction of a new instruction, and the A-operation has as main element the execution of an instruction. However, the distinction between these kinds is not sharp.

The K-digit determines for which unit the short registers are used, i.e. for the arithmetic unit or for the control. Together with the A-digit the K-digit determines the way of coupling between the four parts: arithmetic unit, control, short registers, and drum store. This will be clear from the following diagram:



Cf. the introduction, p. 5.

The function of the Q-digit is the addition of $\pm\varepsilon$ to the B-accumulator, irrespective of the store.

The digits L and R effect the shifting of the contents of the double-length accumulator to the left or to the right, respectively.

The I-digit controls the additive or subtractive action of an instruction. This only applies to the accumulators, not to the control, and then only for the transfer to A or B.

The B-digit determines whether an operation refers to the A or to the B-accumulator. This only applies to adding, not to shifting.

The C-digit determines whether or not the accumulator engaged in the operation must be cleared.

The digits D and E determine whether reading or writing takes place from (to) the drum and the short registers respectively.

The digits V, V_4, V_2, V_1 are called the test digits. With a testing operation the operation is either or not executed, dependent on the criterion described by the digits V, V_4, V_2, V_1. If the instruction is not executed, an instruction AO is executed instead.

The digit W is related to the time selection on the drum. If $c_{14} = 0$, the execution of an operation is delayed till the selected storage location on the drum is present. If $c_{14} = 1$, the operation is executed immediately without the drum being waited for. The drum is completely disregarded. 0 is always read and nothing can be written on the drum.

The remainder of the digits forms the addresses. c_{15} to c_{19} constitute the short address and c_{20} to c_{32} constitute the drum address. c_{20} to c_{27} serve the track selection and c_{28} to c_{32} serve the time selection within the selected track. For the sake of shortness the contents of the drum address will always be denoted by (n) and the contents of the selected short address will always be written as (m). (n) = 0 if the waiting digit is 1. If (n) is destined for A, this number is denoted by $(n)_A$. Then $(n)_B$ and $(n)_C$ are 0.

In the same way by $(m)_A$, $(m)_B$ and $(m)_C$ is denoted the contents of (m) as far as they are destined for A, B or C. Both other entrances receive a 0.

2.4 The action of the instructions

2.41 The functional digits

The A-digit

In the control the A-digit has the following action:

Operation X: $(C) + 2\varepsilon \rightarrow D \qquad (n)_C + (m)_C \rightarrow C$

Operation A: $(m)_C + (D) \rightarrow C \qquad (4) \rightarrow D$

Both operations do not differ in so far as the arithmetic unit is concerned. In any case adding or storing takes place according to:

$$(A) \pm \{(n)_A + (m)_A\} \rightarrow A \qquad (B) \pm \{(n)_B + (m)_B\} \rightarrow B$$

These standard operations can be modified by the other operation digits.

Register 4 has a special function and is related to the A-operations. All instructions are either X-instructions or A-instructions.

The K-digit.

If the K-digit is absent: the short store is used for the arithmetic unit.

If K is present: the short store is used for the control.
On a reading operation: $(m) \rightarrow C$
On a writing operation: $(D) \rightarrow m$

The Q-digit.

If the Q-digit is absent: normal.

If Q is present, ε is added to (B) (or is subtracted dependent on I). The ε is introduced in the carry entrance of the pre-adder of B as if it were a carry from "b_{34}". The adding of ε under control of Q is also taking place on a storing operation.

The L-digit.

If L is absent: normal.

If L is present: (A) and (B) are shifted one place to the left. If A and B are not cleared the leftmost digit of B shifts to the rightmost digit of A, and B is completed on the right-hand side with a 0. The leftmost digit of A is lost. If A or B are cleared, 0 is always transported from B to A. All other operations are performed in the normal way.

The R-digit.

If R is absent: normal.

If R is present: A and B are shifted one place to the right. When A and B are not cleared, the rightmost digit of A shifts to the leftmost digit of B. The rightmost digit of B is lost. A is supplemented on the left-hand side with a digit from a place which will be called a_{-1}. This place is situated on the left side of a_0, and completes the A-accumulator to an adder of 34 places instead of 33 places. For this extra place the following rules hold:

If A is cleared, a_{-1} is also cleared. All numbers to be added are first added together in what is called the pre-adder; then the resulting number is completed with a copy of its sign digit, after which the number of 34 digits is added into A with the main adder. This digit is serving effectively to store an overflow. The only method to recover this digit is to shift it to the right by an R-operation. The shifting to the right prevails over shifting to the left; thus a combination of R and L shifts to the right only.

For the sake of doing multiplications the following facility has been added to LR: If LR is present, add $b_{32}.(15)$ to A instead of $(m)_A$.

The I-digit.

If the I-digit is absent: normal.

If the I-digit is present: take the complement of the numbers of drum and short register, in so far as they are destined for the arithmetic unit. The contents of 15 on an XD and an LR-operation and the ε on a Q-operation are also complemented when I is present. The I-digit does not refer to numbers to be stored, or to the control.

The B-digit.

If the B-digit is absent: the operation refers to A.

If the B-digit is present: the operation refers to B.

The addition normally takes place in A just as the storing normally takes place from A. However, if the B-digit is present, the addition takes place in B and the storing also takes place from B. The B-digit has no influence on the addition of (15) to A on an LR-operation. This addition always relates to A. The addition or subtraction of ε on a Q-digit also always takes place in B. The B-digit has no relation to the control.

The C-digit.

If the C-digit is absent: do not clear A and B.

If the C-digit is present: clear the accumulator as prescribed by the B-digit, before an addition or a shift takes place. The C-digit does not relate to the control.

The D-digit.

If the D-digit is absent, and if the execution is waiting for the drum: read the number from the selected drum storage location and perform on it an operation according to the other digits.

If the D-digit is present, and the execution is waiting for the drum: write in the selected drum storage location the number from A or B according to the following rules:

With an operation without B:	$c_8 = 0$: (n) destined for A.
	$c_8 = 1$: Transfer (A) to n.
With a B-operation:	$c_8 = 0$: (n) destined for B.
	$c_8 = 1$: Transfer (B) to n.

On the combination of X and D an extra addition takes place: Add (15) instead of $(m)_A$ or $(m)_B$ to A or B according to the B-digit.

The E-digit.

If the E-digit is absent: read the relevant short register and use it for A, B or C according to the K and B-digit in the operations.

If the E-digit is present: read the number as determined by K and B in the selected register.

If K and B are both absent:	c_9 = 0: (m) destined for A.
	c_9 = 1: (A) $\rightarrow$ m
If K is absent, B is present:	c_9 = 0: (m) destined for B.
	c_9 = 1: (B) $\rightarrow$ m
If K is present:	c_9 = 0: (m) destined for C.
	c_9 = 1: (D) $\rightarrow$ m

2.42 The test digits

If the V-digit is not present, the digits V_4, V_2 and V_1 together determine a number, having the value 0 to 7. These are indicated by U0 to U7. If the instruction contains Uk, this operation is executed if the selection switch k has been thrown. If not, the operation A0 is executed. The selection switches will be denoted by KS0 to KS7. KS0 is always thrown. An instruction with c_{10}, c_{11}, c_{12}, c_{13} = 0 will be executed in the normal way. KS7 is materialized as a key having a normally closed contact. This key serves as a start key.

If the V-digit is present, a V is added to the instruction.

$c_{10}, c_{11}, c_{12}, c_{13}$	= 1000 is denoted by V :	Cf. 2.43
"	= 1001 is denoted by V1:	Execute the instruction if a_o = 1; if not execute A.
"	= 1010 is denoted by V2:	Execute the instruction if b_o = 1; if not execute A.
"	= 1100 is denoted by V4:	Execute the instruction if b_{32} = 1; if not execute A.

With the aid of these functional digits a test can be performed.

2.43 Double-length facilities

To be able to perform double-length arithmetic very easily a device to take the carry-over from B to A is provided. As this carry-over is only produced on the last impulse time in a word, it is not possible to add it to A in the same cycle. This is always done in a later cycle (not necessarily the next).

The normal rule for double-length arithmetic is as follows: On every B or Q-operation the carry-over is stored in an intermediate storage of one digit. This carry is added to A on the first instruction having a V0, which can be written simply as V. The B-instruction and the related V-instruction must have an equal I-digit. The intermediate store retains the carry which has been put into it on the last B or Q-operation. The carry from the intermediate store is introduced on the carry entrance of the pre-adder of A as if it were a carry from "a_{34}". On a left shifting instruction with V it is introduced one digit time later as if it were a carry from a_{33}. This implies that an instruction of the form A200L5V can give wrong results, because the addition of (200) and (5) in the pre-adder can give rise already to a carry from a_{33} to a_{32} so that no other carry can be added at the same time.

For a better understanding a short account will be given of the precise action of the intermediate store. A subtraction in B is performed by adding the inverse of the number together with introducing an extra complementary one on the carry entrance of the main adder of B as if it were a carry from "b_{34}". When a number is added, the resulting carry is just the opposite of what it would be, when the same number would be subtracted. For example subtracting 0 gives a carry 1. In general this can be formulated as follows: The borrow produced on a subtraction is the opposite of the carry produced by adding the complement. However, on the next V-instruction the fact that a borrow has been stored in the intermediate store in opposite form must be taken into consideration by reversing its significance as a I-operation. The negative value of a borrow is automatically accounted for by the introduction into the pre-adder. The result of this pre-addition (now including the borrow) is subtracted from A on a subtraction.

These seemingly awkward rules are necessary to be able to round-off on multiplication with a special trick, and to use the V as a sort of "Q-digit" for the A-accumulator.

Examples: Round-off on multiplication

N...IB23	Last instr. of mult. contains I. B23 subtracts $\frac{1}{2}$ from tail giving carry-over when tail $\geqslant \frac{1}{2}$.
N...V	Round-off is added to head on next operation. B-instr. and corresponding V-instr. do not have the same I-digit!

The use of V as "Q-digit":

N...LBI	Subtract 0 from B thus making carry = 1
⋮	
N...V	Add extra 1 to head from intermediate carry store etc.

2.44 The order of preference

The functional digits of the operation are written in a certain order. This order is: AKQLRIBCDEVV$_4$V$_2$V$_1$. By reading it from the right to the left the order of preference of the functional digits is given. One can imagine the action to be thus that all functions take place subsequently. First from the test digit it is tested whether the operation is taking place or not. Then if storing has to take place first storing is effected. Then if clearing has to take place the clearing is performed. The relevant register is indicated by the B-digit. The position of the inversion digit is of no importance. The order of LR indicates that R has preference over L. When R and L are used together, only a shift to the right is effective. As last action the additions with Q and A take place. The position of the K-digit is unimportant.

2.45 The notation of the instructions

All instructions begin with an A or an X. As an X jumping to the immediately following register is very frequent, an abbreviation will be introduced: (p) = Xp+1 is to be denoted by N. Then other functional digits still can be added.

In general an instruction consists of an opening symbol A, X or N, after which one or more addresses follow together with functional digits. When a drum address (having 3 digits or being ≥ 32) is written first, the wait digit is kept 0 by the input programme (Cf. 2.83); when the short address comes first, the wait digit W is automatically set to 1. A short address 0 may be omitted completely. Two addresses are separated by functional characters or by a dot. E.g. X123.4 or A356BC5. However, when a short address comes first and when a drum address is following, W = 1 and the drum address is no more effective as an address. With respect to the application of counting, the address when written as g is put in as 8192 - 2g. XK6R31 has W = 1, short address = 6, drum address = 8130.

2.5 Some examples

2.51 A simple programme

Be asked to add $(133) + (135) - (4) \rightarrow 6$ and $(B) - \varepsilon \rightarrow B$

100	AC133	$(133) \rightarrow A$
101		133 can be reached just in time
102	A135	$(135) + (A) \rightarrow A$
103		
104	NQI4	$(A) - (4) \rightarrow A$ and $(B) - \varepsilon \rightarrow B$
105	NE6	$(A) \rightarrow 6$
106	N	Neutralize special jump

When these instructions are given the following is happening in the registers:

A	B	C	D	
a	b	X100 AC133	X102	During A-instructions a continuous

A	B	C	D	
(133)	b	X102		transport of the intermediate X-instr. is taking place between C and D.
		A135	X104	
(133)+(135)		X104		
		X105QI4	X106	
(133)+(135)-(4)	b-ε	X106E6	X107QI4	Here only transport from C → D takes place. (A) → 6
		X107	X108E6	
		etc.		

2.52 The use of the return instruction

One of the aids used most in programming is the calling in of a sub-programme. With this facility all complicated operations such as multiplication, division, square-rooting, calculation of elementary functions etc., can be reduced to one instruction.

The calling-in of a sub-programme (for instance on 200) is effected with X200KE4: jump to 200 and place return instruction in 4. This return instruction may be placed in every other short register according to the requirements. The lay-out of the main programme and the sub-programme is as follows:

100	X200KE4	Jump to 200 and place return instruction in 4.
101		
102	Programme returns here	

Sub-programme:

200	N...	Neutralize KE4
⋮		
200+k	XK4	Jump to return instruction

The actions in the registers are as follows:

A	B	C	D	4
		X100		
		X200KE4	X102	
		N...		X102
		⋮		
		X200 + k		
		XK4	X202 + k	
		X102		
		etc.		

Often a sub-programme which is completely contained in the short registers is called in (sometimes with one instruction only). This is done as follows:

100	NKE7	Place return instruction in 7
101	XK6	Jump to 6 (if necessary with other functional digits).
102	etc.	Programme returns here.

6	Instruction
(7	Return instruction

The property of returning two instructions later on is very useful here.

This latter property is less desirable for normal sub-programmes. Hence it is better to adopt the convention that as a rule all sub-programmes will return on the next instruction (p + 1). The sub-programme then runs as follows:

200	N...	Neutralize
⋮		
200+k	NK4	Take in return instruction + (−ε)
201+k	−1	into C.

2.53 Stopping and starting

A stop instruction as such is unknown in the ZEBRA code. It may be replaced by using the so-called loop-stop: (100) = X100. When the machine comes to 100 the control continuously takes the next instruction from 100. It is a drawback that the programme cannot leave this point of its own accord.

For the following method of stopping the selection switch 7 is used.

100	X100KE4U7	Conditional loop-stop
101		
102	Xp	

The selection switch 7 has been materialized as a key having a normally closed contact, which is opened when being pressed. The following takes place in the registers:

C	D	4	
X100			
X100KE4U7	X102		
X100KE4U7	X102KE4U7	X102	
X100KE4U7	X102KE4U7	X102KE4U7	Now e.g. KS7 is pressed

C	D	4	
X102KE4U7	X102KE4U7	X102KE4U7	X100KE4U7 = A = AO !!
X102KE4U7	X102KE4U7		Also X102KE4U7 = A until
Xp	X104KE4U7		KS7 is released.
(p)	Xp + 2		
etc.			

When KS7 is being pressed nothing happens, but when the key is released, the programme resumes its normal action. This procedure has the advantage that the machine, after having been started, can immediately stop again. When KS7 is kept depressed, the machine may skip a stop instruction of the before-mentioned type.

In order to make the machine start at a predetermined point, another key has been provided which clears C. Now X000 is executed (on purpose the instruction which completely consists of zeros has been chosen here). Normally at 000 a stop instruction X000KE4U7 may be found.

2.54 The use of the test instructions

A test can occur with two types of operations: with X-operations and with A-operations.

With a testing A-operation adding or storing takes place dependent on the tested criterion. Frequently in the course of a programme bifurcations occur. In that case the test is attached to an X-operation, which either jumps or continues on the instruction 2 registers further on.

100	X200V1	Jump to 200 if (A) $<$ 0
101		
102	etc.	Otherwise go to 102

It is very important that on 200 first the special jump X200V1 is neutralized by another jump, as otherwise the next instruction would be taken in by X202V1, etc. Testing would then take place continuously on the intermediate jumps. When the programme is beginning on 200 with N....., the former jump is automatically neutralized.

A few special applications of testing instructions are:

100	AIC2V1	Form modulus of (A)

and

100	AIBC3V2	Form modulus of (B)

2.6 Possibilities of the code

2.61 The repetition of an instruction

In ZEBRA there is a possibility to repeat an instruction which is located in a short register, a number of times (maximum 4096 times). In general this is performed in the following way: (The examples given below all begin at 100. This is, however, only arbitrary.)

100	NKE7	Place return instruction in 7
101	X6K...p	Jump to 6 and repeat p times
102	etc.	

6	instruction to be repeated
(7	return instruction

Here p is the number of times that the instruction must be repeated. The instruction on 100 places the return instruction on 7, and then continues on 101. Here is to be found: jump to 6, while the drum address is 8192 - 2p which, as an address, has no effect. If necessary, other functional digits may be added. The drum address in the X-instruction is augmented by 2 each time when it is transported from C to D; hence after p times the drum address is equal to 8192. But this is exactly one unit of the short address, which consequently directs the machine to the next short address. Here the return instruction has been placed, so that the repetition of the instruction is terminated. It is not necessary to use 6 and 7 for the repetition; 3 and 4 may also be used. The repetition jump need not be in 101 either; it may be placed in any arbitrary place. It is also possible to repeat the calling in of a sub-programme. The instruction to be repeated may read as follows: XnKEm. In principle any number of instructions from the drum can be repeated with this trick.

2.62 The pre-instruction

It often happens that a constant must be placed in a short register while effecting an N-operation. This can be done as follows:

100	NE5	Store in 5
101	AC102	(102) ⟶ A. X103E5 goes to D !
"102	constant	
103	N...	This instruction is taken in with
104	etc.	X103E5, which effects the storing in 5.

In this example the storing is attached to the X-operation. As this X-instruction is always circulating between C and D, the storing is also effected after taking in the constant.

In general much attention must be paid to the use of an

A-instruction after a special X-instruction.

In all examples the programme is always assumed to begin with a normal jump. This can always be done by an N on the last instruction.

2.63 Multiplication

The multiplication of two numbers with sign is done according to a system devised by VON NEUMANN. The multiplicand is put into short register 15. The multiplier is put into B, and A is cleared. Dependent upon the extreme right-hand digit of B, the contents of 15 are either or not added into A. Then a shift to the right takes place. B thereby loses a digit on the right-hand side, but receives a digit of the tail of the product on the left-hand side. (This digit is not altered any more in subsequent action.) In B the multiplier disappears gradually, while the tail is shifting into B. The head remains in A.

With the LR-facility the nucleus of the multiplication runs as follows:

100	NKLRCE5	Place return instruction in 5
101	X4K15LR	Repeat (4) 15 times (X4KLR is done 16 times)
102	NLRI	Treat last digit negatively

4	ALR
15	Multiplicand

According to the complement representation for numbers the last digit of the multiplier must be treated negatively. For example:

$$a \times (-\tfrac{5}{8}) = a \times 1.011 = a \times (1.000 + 0.011) = a \times (-1 + \tfrac{3}{8})$$

The execution time is 11 ms. ALR has to be placed in 4 in advance but this can be done once for all when many multiplications must be done (e.g. in power series etc.).

A simpler way for multiplication is the use of a sub-programme.

An example of a general sub-programme for multiplication will be given here. When called in with X100KE4Q it determines (A).(B) in double length, head in A, tail in B. When called in with X101KE4Q it determines (A).(B) rounded off in A. B contains tail + ½. Suppose that at the beginning: (A) = a, (B) = b

	100	X102E5	Pre-instr. (B) = b + ε. Plant carry 0
	101	NE5V	No carry
100→	102	A103CE15QI	a→15. Plant carry = 1. Restore (B) = b
"	103	-ALR	ALR→5 with pre-instruction. V active but harmless

104	NKE6LRC	Place return instruction. Clear A
105	X5K15LR	Repetition for multiplication. V, if present, adds round-off on penultimate cycle.
106	NK4LRI	Last cycle negatively. Return to main programme.
"107	-1	Constant for decreasing return instr. by ε.

When a multiplication must be done with a constant factor, it can be done faster than the general multiplication programme, when the constant factor has less than 33 digits. Suppose that (5) must be multiplied by 0.1011. A and B are cleared at the beginning.

100	N5	Form (5) x 1.0
101	NR5	Form (5) x 1.1
102	NR	Form (5) x 0.11
103	NR5	Form (5) x 1.011
104	NR	Form (5) x 0.1011

With this example the method has been elucidated sufficiently. Head and tail of the product are available. The time of operation can be much less than 10 ms.

The same method can be applied for multiplying from left to right, e.g. (5) x 101101.ε :

100	NBC5	Form (5) x 1ε
101	NL	Form (5) x 10ε
102	NLB5	Form (5) x 101ε
103	NLB5V	Form (5) x 1011ε
104	NLBV	Form (5) x 10110ε
105	NLB5V	Form (5) x 101101ε
106	N...V	V for last carry-over

2.64 The division

Division is the inverse operation of the multiplication. Hence it must be possible to divide a double-length dividend by a single-length divisor to obtain a single-length quotient and a single-length remainder. The system that has been used, is the restoring division. Only the case of division of non-negative numbers is of importance. According to VON NEUMANN the double-length dividend will be put into A and B (head in A, tail without sign of its own). The divisor will be put into 15. After the subtraction of the divisor in A it can be tested from the sign whether the subtraction was possible or not. The digit of the quotient may then be placed in the last (non-occupied) place of B; if necessary the divisor is again added if the subtraction was not possible. Ultimately A and B together are shifted one place to the left. From B a fresh digit is shifted to A, while in B a place is available to take the next digit of the quotient. At the end of the operation the remainder is in A and the quotient in B.

In order to programme the afore-mentioned process it will be split up into two parts:

a. Shifting and subtraction of the divisor.

b. If necessary, noting down the digit of the quotient and the addition of the divisor.

The operation a takes place in any case, the operation b, however, depends on the sign of (A) after the subtraction. If (A) becomes negative, the subtraction must be annihilated, whilst no digit of the quotient is added. If (A) remains positive, the subtraction should not be annihilated, whilst a 1 should be added to the quotient. This difficulty can be avoided by adding the negative of the divisor and at the same time adding a 1 during period a. During period b both the subtraction and the addition of a 1 in the quotient are either or not annihilated.

As the testing is always performed on the signs, the dividend and divisor should be made positive beforehand and the sign of the quotient should be taken care of afterwards. A slight advantage is that, as the quotient is always positive, the first subtraction has to take place in no case.

The nucleus of the division is performed as follows:

(A) = head of the dividend with sign (= 0)

(B) = tail of the dividend without sign, directly following the head. $b_{32} = 0$

(15) = - divisor

100	NKE7	Place return instruction in 7
101	X6K31LDQ	Shift, subtract divisor and add a 1 to the quotient. Repeat instruction on 6 31 times.
102	AQI15V1	The last operation of the repetition is shifting. Hence the 32nd restoration must still be performed.
103		
104	etc.	

6	AQI15V1
(7	return instruction
15	divisor

When fractions are divided a rounded off quotient is mostly desired. This rounding cannot be performed by adding $\frac{1}{2}$ to the quotient, as the quotient is only of single length. Hence the rounding is effected by augmenting the tail of the dividend by $\frac{1}{2}$ x (divisor). For this purpose it is, however, not necessary to form half of the divisor. As the tail has no separate sign digit, the sign digit of the divisor effects the right spacing between the tail and the head of the dividend.

The general division programme for (A)/(B), rounded off to A and B is running as follows:

100	NIBC3	Set borrow = 1. NIBC3 can act two times
101	X103IC2V2	Form $-\lvert b\rvert$ and complement a accordingly
102		
101→103	NBE15IV	Store $-\lvert b\rvert \rightarrow 15$ $\quad a - \varepsilon \rightarrow A$
104	NE6	Pre-instruction
105	A106CE7R	$a - \varepsilon \rightarrow 7$. R on 105 and LQ on 107 are setting $b_{32} = 1$
"106	AQI15V1	
107	A108CE5LQ	AQI15V1 →5
"108	X114I15	X114I15 →6 as a return instruction
109	NC7	$a - \varepsilon \rightarrow A$. $-\varepsilon$ serves as head for $-\frac{1}{2}\lvert b\rvert\varepsilon$ in tail
110	X112QI15V1	First cycle of division. In fact:
111		$\dfrac{a + b - \frac{1}{2} b\varepsilon}{b} = 1 + q - \frac{1}{2}\varepsilon$
110→112	X5K32LDQ	Repetition for division. Does one cycle to much. X114I15 and R on 114 corrects for this.
113		
6 →114	NRQB23	Form $(1 + q - \frac{1}{2}\varepsilon) + (1 + \varepsilon) = q + \frac{1}{2}\varepsilon =$ rounded quotient. (23) = -(23) !!
115	NC3	Place quotient in B too
116	NK4	Return to main programme
"117	-1	

The time for the actual division is 22 ms. The complete programme has an average operation time of 36 ms.

2.65 Shifting

When a number has to be shifted to the right this can be done most quickly with a number of NR-instructions.

When shifting over a large number of places is necessary, it can be effected better in the following way:

100	NKE7	Place return instruction
101	XK6Rp	Repeat the shift instruction p times.
102	N or NR	

6	AR
(7	return instruction

During the repetition shifting is effected over 2p + 1 places. When the required number of places is even, an NR can be added. The time is 10 ms.

In the afore mentioned examples A and B always shift together. When only A has to be shifted, this can be done by means of NRBC3 or ARBC3. B then receives no carry-over digits from A. In the same way B alone can be shifted by NRC2 or by ARC2. B is now supplemented with zeros.

The shifting over a number of places, which number is itself a result of a previous calculation, is effected as follows:

2	number to be shifted
3	- (number of times)
4	return instruction
5	AQRBC3

100	NKE4	Place return instruction in 4
101	XK5V2	Repeat the instruction in 5 and test
4→102	N	
101→103	etc.	Programme comes back here directly when (3) = 0

As the action has certain peculiarities a diagram of the action in A, B, C and D is given:

A	B	C	D	4
a	-2ε	X100		
		X101KE4	X102	
		XK5V2	X103KE4	X102
		AQRBC3	XK5V2	
$\frac{1}{2}$a	-1ε	XK5V2	X102	
		AQRBC3	XK5V2	
$\frac{1}{4}$a	+0	XK5V2 = A	X102	
		X102		
		etc.		

When B becomes positive, XK5V2 is no longer executed, but instead A (= AO) is executed. Then X102 comes into C and the programme returns. This will be called a testing repetition. For such a testing repetition the return instruction must be placed in 4.

The methods available for shifting to the left are analogous to those for shifting to the right. Only LR must be dealt with carefully.

The same method used with R, for shifting only one of the accumulators, can be used with L. However, it is better and easier to use N2 or NB3, viz. add A (B respectively) to itself.

2.66 Normalization

Normalization is the shifting of a number to the left until it lies between $\frac{1}{2}$ and 1, together with the counting of the number of places shifted. This operation is important with regard to floating point arithmetic.

Suppose that the negative number a is in A.

100	NKE4BC	Place return instruction in 4 and 0 → B
101	XK5V1	Repeat AQ2 (shift A, count in B) until the test tells that the first zero in A has come in a_0
102	etc.	

4	return instruction
5	AQ2

The time is 10 ms or 20 ms. The same programme can also serve to search the leftmost 1 in a number. Instead of a normal count a shift count is then held in B by beginning with ε in B and making a double-length shift by (5) = AL

2.67 Automatically variable instructions

With ZEBRA it is possible to operate with instructions which, although they are fixed in the store, yet are modified in the address with a variable count before they are executed. Suppose that the number must be taken in from register 300 + k.

100	NK6	Take in AC300 + kK7 → C
101	AC300K7	Execute the operation and form
102		X103 in C
103	etc.	

6	k
7	– X000K6

The action diagram is:

A	C	D	
	X100		
	X101K6	X102	
	AC300+kK7	X103K6	
(300 + k)	X103		(7) is added to C

It often happens that a number must be looked up in a small table. This is e.g. the case with recoding. Suppose that the digits 0 - 9 must be recoded. Be the digit k, then the recoded digit is denoted by f(k). With the application of the afore mentioned method for looking up, a complete revolution would be needed. However, by doing it in the following way it can be effected with a minimum of access-time. Suppose (A) = k

100	NK2	Take in AC102+kK6 ⟶ C X103K2 ⟶ D
101	AC102K6	Take in f(k)
102	f(0)	X103K2 - X000K2 + X009 = X112 ⟶ C
103	f(1)	
104	f(2)	(6) = - X000K2 + X009
105	f(3)	
106	f(4)	
107	f(5)	Function table
108	f(6)	
109	f(7)	
110	f(8)	
111	f(9)	
112	etc.	Programme resumes its action here.

The time for looking up is always equal to the length of the list. In some cases this process can be speeded up considerably by placing 32k in A. With this trick the table is spread out horizontally on 102 + 32k. All f(k) can be reached in the shortest possible time. This is called a horizontal list or ladder.

It often happens that a jump to a variable place must be executed. This can be effected as follows:

100	NK2	The jump instruction which is taken in, neutralizes the special jump NK2, so that no special constant in 6 is needed
101	N	Becomes X102 + k
102	Xa	
103	Xb	
104	etc.	

This ladder of jumps can also be spread out horizontally.

The execution of variable instructions which are dependent upon two counts can be effected with the following programme: Suppose that we want to look up a number in 300 + k + 1

100	NC4	Form k + 1 in A
101	N5	
102	NK2	
103	AC300K6	Execute variable instruction
104		
105	etc.	

4	k
5	1
6	- X000K2

The only time which plays a rôle in these variable instructions is the access time for an arbitrary number. All other instructions are optimum.

2.68 Block transport

Data can in a simple manner be quickly transported from the drum to the short store and vice versa. The locations on the drum must be placed on k, k + 2, k + 4, etc. With a view to executing every second instruction this must be called the "normal" placing.

A programme which transports 5 numbers from m, m + 2, m + 4, m + 6, and m + 8 to 6, 7, 8, 9, and 10 runs as follows:

100	NC5	(5) → A
101	ABC102	
"102	X002.1	
103	AB104CE15	X002.1 → 15
"104	AmCE5	AmCE5 → B
105	NKE4	Place return instruction
106	X3K6BD	Repeat (3) 6 times and advance it every time
4 → 107	etc.	

As this programme uses the XD-facility, an action diagram will be given.

A	B	C	D	
		X100		
		X101C5		
(5)		ABC102	X103C5	
	X002.1	X103C5		
		AB104CE15	X105C5	X002.1 → 15
	AmCE5	X105C5		
		X106KE4	X107C5	

A	B	C	D	
(5)	AmCE5	X3K6BD		X107C5 → 4
	Am+2CE6	AmCE5	X3K5BD	
(m)		X3K5BD		(5) → 5
	Am+4CE7	Am+2CE6	X3K4BD	
(m+2)		X3K4BD		(m) → 6
	Am+6CE8	Am+4CE7	X3K3BD	
(m+4)		X3K3BD		(m+2) → 7
	Am+8CE9	Am+6CE8	X3K2BD	
(m+6)		X3K2BD		(m+4) → 8
	Am+10CE10	Am+8CE9	X3K1BD	
(m+8)		X3K1BD		(m+6) → 9
	Am+12CE11	Am+10CE10	X4KBD	
(m+10)	.	X4KBD		(m+8) → 10
		X107C5		
(5)		etc.		

Once m has been reached, all following instructions are just in the right places.

A second way of doing block transport is by stretched programme. A fine example is the following sub-programme of one trackful of instructions (including working registers) to transport the contents of all the short storage locations to the drum and back with the same set of instructions!

It is called in with: NC4
X100KE4BC
to transport from short store to drum. It is called in with:
X100KE15BCQ
to transport back from drum to short store.

		SS → DS	DS → SS
100	NK3	(B) = 0 (A) = (4)	(B) = 1
101	AIBC101V4	Test fails	Acts as AIBC102!
"102	X000E2	(B) = still 0	(B) = –X000E2 < 0
103	AD104C5	(4) → 104	AD104C5–X000E2 = A104CE3
(104			(104) → A
105	AD106C6	(5) → 106	A106CE4 → 0
(106			(104) → 4
107	AD108C7	(6) → 108	(106) → 5
(108			
109	AD110C8	(7) → 110	(108) → 6

	(110			
	111	AD112C9	(8) → 112	(110) → 7
	(112			
	113	AD114C10	(9) → 114	(112) → 8
	(114			
	115	AD116C11	(10) → 116	(114) → 9
	(116			
	117	AD118C12	(11) → 118	(116) → 10
	(118			
	119	AD120C13	(12) → 120	(118) → 11
	(120			
	121	AD122C14	(13) → 122	(120) → 12
	(122			
	123	AD124C15	(14) → 124	(122) → 13
	(124			
	125	AD126C16	(15) → 126	(124) → 14
	(126			(126) → A
	127	NBCE17V2	Test fails	Becomes: NBC15V2: (15) → B
	128	X130E15		(126) → 15
127	129	NB4	Return instr. → B	
128	130	NK3	Return to main programme	
	"131	-1		

This programme can be used to extend the usefulness of the short store. The drum acts as a backing store. When called in from 99 + 32k, the operation time is only 10 ms.

2.69 The adding of the contents of consecutive registers

With the trick mentioned in 2.68 of placing an instruction in B, the contents of a large number of consecutive registers can be added very quickly. For example: add all registers from 200 to 300:

	100	ABC101	Place AQ200 in B
	"101	AQ200	
	102	NKE4C	Place return instruction
	103	XK3Q51	Execute AQ200+2k 51 times. k=0(2)100
4 →	104	ABC105	
	105	AQ201	The same for the registers having odd numbers
	106	NKE4	
	107	XK3Q50	

With the exception of the access time for the first register, this programme is completely without waiting times. With this programme a check can be very quickly made on a programme which is put in by the input programme by summing up all the registers put in. It is possible to add to every programme a check number in such a way, that the sum is exactly 0. This gives a very quick and reliable check on the input.

2.7 The duality between A and X-operations

It has been shown that programmes only consisting of adding jumps are very practical. It is, however, also possible to make programmes only consisting of jumping additions.

In the first place it is possible to have two A-instructions in succession. An example will be shown of the manner in which a number pair may be taken in from m and m + 1, without waiting time for the second component. (B) = m

100	NE5	Pre-instruction
101	A102C3	ABCm+1K1 – X107K3 → A
"102	ABC001K1–X107K3	Store this in 5
103	NKE4	Store return instruction X105E5 in 4
104	NK3	Take ACmK5 into C
105	AC000K5	Next, ABCm+1K1 is coming to C
106	etc.	

The action diagram is:

A	B	C	D	
	m	X100		
		X101E5		
		A102C3	X103E5	
ABCm+1K1–X107K3		X103E5		ABCm+1K1–X107K3 → 5
		X104KE4	X105E5	
		X105K3		X105E5 → 4
		ACmK5	X107K3	
(m)		ABCm+1K1	X105E5	(4) → D
	(m+1)	X106E5		K1 adds 1 to C
		etc.		

Here the interruption of the alternation X-instruction —— A-instruction plays an active rôle.

The use of a repeated A-instruction without intermediate jumps will be demonstrated in the next programme, which is intended to clear the store from 32 onward.

32	NCE4	Clear 4 and A on second action
33	ABC34	
"34	ACD33K3Q - X39CE4	
35	AB36CE5	$(34) \rightarrow 5$
"36	ACD34K3Q	$(36) \rightarrow B$
37	ACD32K5	Clear 32 and start programme

The action is:

A	B	C	D	
		X33CE4		
0		ABC34	X35CE4	
	(34)	X35CE4		$0 \rightarrow 4$
		AB36CE5	X37CE4	
	ACD34K3Q	X37CE4		
		ACD32K5	X39CE4	
		ACD33K3Q	0	$0 \rightarrow 32$
	ACD35K3Q	ACD34K3Q		$0 \rightarrow 33$
	ACD36K3Q	ACD35K3Q		$0 \rightarrow 34$
	ACD37K3Q	ACD36K3Q		$0 \rightarrow 35$
	⋮	⋮		
	ACD000K4Q	ACD8191K3Q		$0 \rightarrow 8191$
		ACD000K4Q		
		X000		Stop on 000

This type of process is double as quick as a normal repetition instruction, but it can only run to the end of the store.

By having our instruction AmK3Q in B, the sum of all registers in the store can be formed. By attaching a test V1, it is possible to end the repetition of AmK3QV1. This is useful for a ladder test in which a quantity is sorted according to certain group boundaries stored in consecutive registers. As soon as the test fails, the repetition stops by taking (4) as the next instruction. The last contents of B give indication, where the repetition stopped.

It is clear from what has been said, that there exists a complete duality between an adding jump and a jumping addition. On an adding jump the drum store is used in the control, the short store is used for the arithmetic unit, and $(C) \rightarrow D$. On a jumping addition the short store is used for the control, the drum store is used for the arithmetic unit, and $(D) \rightarrow C$. There exist programmes, consisting of X-instructions only, as well as of A-instructions only.

2.8 Input and output

2.81 The input

The taking in of data is performed by means of normal teletype tape. This is read by the aid of a Ferranti optical tape reader, reading 200 symbols a second. Every symbol consists of 5 holes. In order to simplify the technical devices as far as possible, the 5 holes of the tape are read separately and are combined by means of a programme. The registers 26 to 30 serve this purpose. When these registers are read, they indicate a number consisting of zeros if the relevant hole in the tape is absent, and they yield a number consisting of ones ($-\varepsilon$) when the hole is present.

(26) = 5th hole of the tape.

(27) = 4th hole of the tape.

(28) = 3rd hole of the tape.

(29) = 2nd hole of the tape.

(30) = 1st hole of the tape.

A symbol may be read by means of the following programme:

100	NIBC26	Read 0 or $+\varepsilon$ if 5th hole is 0 or 1
101	NLIB27	Read 4th hole and add it to the shifted accumulator
102	NLIB28	Read 3rd hole
103	NLIB29	Read 2nd hole
104	NLIB30	Read 1st hole
105	N31	Step the tape

The transport of the tape to the next symbol is performed by selecting register 31. For this stepping at least 5 ms are necessary.

It is obvious that two symbols per revolution can be read. The testing of input indications if any can be done within the remaining 12 operations. In this way purely binary tapes can be read at a speed of 25 words per second. One word contains 33 bits. 7 symbols of 5 bits would be sufficient. Two more bits are then available for special markings. Hence in 70 ms 14 symbols (equal to 2 words) may be read. These 2 words must then be stored, which takes 11.25 ms. In about 80 ms 2 words can be taken in.

For the taking in of instructions in the normal form of. the input programme 2.83.

2.82 The output

For the printing of results a normal teletypewriter is available in the first place. This typewriter is controlled by a signal in the 7-unit code. The printing of a symbol is started by 20 ms low voltage, then 5 periods of 20 ms each follow forming the symbol to be printed, and finally the printer is stopped by a 30 ms high voltage signal. The code of the 5-symbol units has nothing to do with the binary system as such.

The recording and the time division necessary for typing a symbol is completely effected by the programme. The machine contains only one communication channel to the teletypewriter via register 25. When 25 is selected, the teletype signal is made equal to the sign digit of A. Furthermore (25) = (A). The use of this register will be elucidated by some examples.

The printing of a fraction consists of repeated multiplications by a factor 10, typing the head and retaining the tail for the next multiplication. To this purpose a sub-programme has been made, multiplying a given number by 10 and typing one digit.

This programme is called in as many times as there are digits to be typed. Another sub-programme determines and types out the sign of the number under consideration and forms its modulus. Other programmes, which will not be given here, make it possible to type other symbols such as space, carriage return, etc.

An important problem in these type of programmes is the exact timing. It is necessary that independent of the place where the digit typing programme is called in, the typing is effected at the accurate speed and without losing time. This is effected in the following way. The digit typing programme begins and ends:

100	N...	
⋮		Perform the necessary instructions
---	X131K4	Jump to return instruction
⋮		
"131	-1	Constant for return instruction

Assume this programme to be called in from 162, then further from 163 and finally from 164. The calling in and the return are in a favourable or an unfavourable situation according to the following scheme:

	calling in	return
162	favourable	unfavourable
163	favourable	favourable
164	unfavourable	favourable

In one case only the returning and the calling in are in a favourable position. The time elapsed between the processes of two sub-programmes called in consecutively, is now always composed of the returning from the former process and the calling in of the next process. From the table it is clear that this is always favourable + unfavourable = 10 ms, hence always an equal time. The deconversion and typing programme runs as follows:

On entering the programme, B contains the number to be typed out (positive). When returning, one digit has been typed out, and the residue is to be found in B. The programme is called in with X100KE4C. The contents of 5 are made A102.25 by the sign programme.

100	NLBE6	Store (B) in 6. Form 2(B)
101	ACE25	Clear A. Set teletype signal low
"102	+0	Const. LBE6 still active: 2(B)→6
103	NLB6	Form 4(B) Form 8(B) + 2(B) = 10(B) = (B)'
104	NLBV	Shift and put carry to 0. Form 2(B)'
105	A106CE6R	Store digit to be typed in 6. Form (B)'
"106	X009-X000K6R	LBV still active. Form 2(B)' and 2(106)
107	NK6R	Form (B)' and (106) in A
108	AC109K2	Execute AC109+dK2. Neutralize K6R
109	f(0)	
110	f(1)	Table of teletype equivalents of digits
111	f(2)	0 – 9
112	f(3)	
113	f(4)	For example:
114	f(5)	
115	f(6)	$f(6) = 0.\underbrace{11}_{1}\,\underbrace{00}_{0}\,\underbrace{11}_{1}\,\underbrace{00}_{0}\,\underbrace{11}_{1}\,\underbrace{11}_{stop}$ etc.; 0 = start
116	f(7)	start 1 0 1 0 1 stop
117	f(8)	
118	f(9)	
119	NKE6	Store return instr. to 121 in 6
120	X5K13	Repeat A102.25. This is executed once per 10 ms owing to the harmless constant in 102.
121	X131K4	Return. Placing of $-\varepsilon$ takes care of exact timing
⋮		
"131	−1	(5) = A102.25 placed by sign programme

By the repeated execution of A102.25 the teletype signal is each time made equal to a_o, and moreover A is shifted, so that all consecutive digits come to the place a_o. The following programme precedes the typing out of digits: type sign of (B) and form $|(B)|$.

132	NE5	Pre-instruction
133	A134CE25	The moment of preparation of start coincides with 101
"134	A102.25	A102.25 → 5

```
     135 | N              Neutralize
     136 | NIBC3V2        Form |(B)|
     137 | X141           If (B)<0: jump to 141
135→138  | AC139          If (B)≥0:
    "139 | f(+)           Take teletype equivalent of +
     140 | X119           Type symbol
136→141  | AC142          Take teletype equivalent of -
    "142 | f(-)
     143 | X119           Type symbol
```

It is envisaged to connect also a high speed punch to the machine. Teletype Inc. supplies punches which are able to punch 60 symbols a second and which may be used on 50 symbols per second here. For this punch the separate holes of the symbol should be set by writing in 26 to 30:

Write in 26: set 5th hole equal to a_o

Write in 27: set 4th hole equal to a_o

Write in 28: set 3rd hole equal to a_o

Write in 29: set 2nd hole equal to a_o

Write in 30: set 1st hole equal to a_o

Writing in register 31 takes care of the punching. The punching of a symbol that is leftmost in A is executed by:

```
100 | NLE26
101 | NLE27
102 | NLE28          Set the 5 holes
103 | NLE29
104 | NLE30
105 | NE31           Punch symbol
```

This operation may only take place once per 20 ms.

2.83 The input programme

When the store of the machine is completely cleared, the machine is not able to take in programmes. For this purpose an input programme is necessary. The structure of the input programme determines the convention underlying the notation. To a certain extent the programmer is free in choosing his own conventions.

The notation followed up to here is an attempt to make the notation of the instructions as simple as possible. The input programme can test from the number of digits whether the address is a drum address or a short address, and dependent on the order of the addresses a wait digit can be added. The abbreviation N for Xp+1 also saves much writing and punching.

In this thesis use has already been made of the code as it will be read by the input programme. However, a few important points are missing:

1. The input indications which tell the machine where to put in programmes into the machine.
2. The input of numbers.
3. The use of parameters to make the relative input of programmes possible which is necessary to make the tapes for sub-programmes general with respect to the place where they are coming in the store.

The most important demand to be made on the code is the necessity that the machine must have a clear criterion indicating what must happen (for example the end mark of an instruction). This is done by dividing the symbols into opening symbols and supplementary symbols. Only X, A, N, +, –, and T are opening symbols, all other symbols are supplementary ones. The opening symbols also play the rôle of end symbol of the former instruction. Hence "nothing" must also be denoted by "something" (viz. X). The separation of both addresses must be marked too. When apart from X, A or N no further supplementary symbols are present, this separation must be done by. On the number of digits the input programme can test whether it has to do with a drum address or with a short address and also which address comes first.

In order to begin the input in a certain place, input indications are necessary. Two kinds are possible:

a. Begin putting in at ...
b. Start executing at ...

These actions will be denoted by the symbols Y and Z. These symbols will be called the input indication symbols. They are attached to a normal instruction.

The storing on the drum of instructions taken in consecutively is done in ascending order. With the input indication pY, the store instruction is replaced by the instruction p. Normally this replacing is done with an instruction ADn. Hence:

ADnY: begin input at n

It is also possible to replace the store instruction by other kinds of instructions, e.g. by Xn. In this way we could leave the input programme. The disadvantage of this method is that the indication where the last instruction has been put in is lost. As it is an advantage that the store instruction is retained, an independent method to leave the input programme is used.

When a Z is attached to an instruction, the action of pZ is: execute the instruction p and proceed with input. In this way the input programme can be left with XnZ. But often it is useful to effect other instructions, e.g.:

AnZ: add (n) to the accumulator and proceed with input. However, this is not possible in the real accumulator, as A is in use for the input itself. Hence such a Z-instruction is executed in the following way:

n	AC4	Take in (4)
(n+2	....	Execute the Z-instruction
n+4	NE4	Store result in 4

Register 4 acts as a phantom accumulator. It is clear that only instructions without B are admitted for having a Z attached.

Example of a programme on the tape:

..	...	
..	...	
	AC200Z	
	A7Z	Place (200) + (7) → 8
	AE8Z	without interrupting further input
..	...	
..	...	

The symbols Y and Z have a second important function. They serve as a special end symbol. After Z the instruction is definitely closed, while all normal instructions are ended by the opening symbol of the next instruction. When the tape is ended with AZ the following parts may begin with blanks on the tape. The O (= blank) is no opening symbol and as the former instruction has been ended, it can be no normal supplementary symbol. So each of the supplementary symbols can be used as a special opening symbol. With a special opening symbol O blank tape can be skipped. The condition in which the machine comes when started on 000 is the same as after Y or Z. The other special opening symbols are still free.

During input any special operation can be effected by means of XnZ by jumping to the relevant part executing this special action. This programme can return by itself to the input programme and resume input.

The tape can be stopped by giving XZ. The action is:
Execute X. (O) = X000: jump to 000. (000) = X000KE4U7: stop.

The input of numbers can be effected most easily with a code which resembles the normal writing of numbers as much as possible. Therefore:

Integers: e.g. +345 Maximum 9 digits. Zeros at the left may be deleted. The point is on the right.

Fractions: e.g. -.345 or -0.345 Maximum 9 digits. Plus and minus are the opening symbols here and . is a special supplementary symbol to prepare the conversion from integer to fraction.

The opening symbol T is used for floating addresses and will not be discussed further.

2.84 Parameters

A very important facility of the input programme is the possibility to modify instructions during the input, e.g. to make programmes independent of the place where they are entering the store. For this purpose it is necessary that one or more parameters can be attached to each instruction.

Parameters are always considered to belong to drum addresses. Therefore they always have to be written directly before the drum address. This must be done in the following way:

pPa

where p is the number of the parameter, P denotes that p is a parameter number and a is the drum address that can be written in this case with fewer than 3 digits.

The significance is:

pPa = a + (p)

p can be a short address or a drum address. This is again normally indicated by fewer or more than 2 digits. For example:

A8PL13 = A000L13 + (8)

X200P5RC15 = X005RC15 + (200)

The input programme is able to accept more than one parameter in a word. In this case these parameters are cumulative. The significance is:

pPqPr = ((p) + q) + r

With this artifice a set of parameters belonging to a sub-programme can be written in the sub-programme itself. Only one parameter is needed to tell where the programme begins, and where the other parameters are placed.

As it is always necessary to use the P directly before the drum address, no confusion can result between:

A5P6C7 = A006C7 + (5) and

X5K6P7 = X5K{7 + (6)}

where the drum address is in fact equal to 8192 - 2{7 + (6)}.

Addresses with a parameter can be looked upon as a whole and can be written before the margin as such.

	AD9PY	Begin to put in at 9PO
9PO	NE5	N is automatically equal to X9P1
1	A9P2C	
"2	X8P	Constant = (8)

9P3	X33P	Jump to sub-programme of which the call-in combination is in 33.
4		
5	etc.	
	⋮	
	X9PZ	Begin to execute at 9P0

The parameters mainly find an application in sub-programmes. The following organization can be very useful.

Every standard sub-programme has a number, for example from 40 onward. The sub-programmes necessary for a problem are consecutively fed into the machine and they are also written consecutively in the store, because every programme marks the end of itself as the beginning of the next programme in 9. The programme too, notes its own place in the register denoted by its own number. For example programme 154:

	AD154Y	Put on 154 the instruction which must
154	X9PKE4	be used for calling in 154.
	AD9PY	Begin to put in programme on the regis-
9P0	------	ter given by (9). The true address con-
1	------	tained in (9) need not be known to the
2	------	programmer.
	etc.	

This method has the advantage that it is not necessary for the programmer to know on which address the return instruction must be put (this need not always be 4). In this manner the calling in of programme 154 is done with X154P.

Perhaps it seems a little wasteful to reserve a number of registers exclusively for this directory. These registers, however, can be used as working registers in the rest of the programme, once the programme has been fed into the store.

2.85 The code on the tape

All symbols used have a character on the tape. There are 32 possible combinations:

0	0	
1	1	
2	2	
3	3	
4	4	Numerals
5	5	
6	6	

7	7	
8	8	
9	9	
10	K	Short store for control
11	Q	Qount, Quotient
12	.	Decimal point
13	L	Left shift
14	R	Right shift
15	I	Inversion
16	B	Use B-accumulator
17	C	Clear
18	D	Store in Drum
19	E	Store in short register
20	T	Floating addresses
21	U	Selection switch
22	V	Test
23	N	Next instruction. Neutralise
24	A	Add
25	X	Jump
26	+	
27	-	
28	Y	Begin to put in on ...
29	Z	Execute on ...
30	P	Parameter
31	erase	Correction

The last symbol is very well suited for correction because it can be punched over all other symbols. After a correction the whole word must be repeated.

2.86 The pre-input programme

When the store of the machine is completely cleared, the machine is not able to take in data from outside the machine. In order to bring the normal input programme into the machine it is necessary to put a minimum input programme into the machine with the manual keyboard. This minimum input programme is called the pre-input programme. It runs as follows:

6 →	000	X5K33	Repeat (5) 33 times
4, 5 →	001	N2	Double (A)
	002	NI26	Read one bit from the tape

003	X3K1	Jump to 3 and execute one time
003 → 3	AnD31	Store instruction. Step tape
(4		Return instruction X5Km
4, 000 → 5	X001KE4	Jump to 001 and store return instr.
4 → 6	X000QI	Jump to 000 and decrease store instr.

When C is cleared by pushing the clear-button, the machine automatically begins on 000. A jump to 5 is executed and the instruction found there is repeated 33 times. On 5 there is an instruction calling in a sub-programme in 001, 002, 003, 3, and 4. The sub-programme is used to take in one bit of the tape at a time. After 33 operation cycles a whole word is built up. Every partial result is stored with the instruction AnD31 in n. After the contents of n have been completed, (6) returns to 000 at the same time subtracting 1 from the store instruction in 3. In this way the normal input programme can be put in in a reverse sequence. The programme ends its action, because, after putting in 004, 003 is also destroyed. This is done in the following way. On 004 half the word which has to come on 003 is put in. Then when the programme begins to build up the contents of 003 this is effected by doubling the contents of 004 and adding one digit. When 003 is passed after taking in this first digit, the contents of 003 are also replaced by the word which has to come to 003. On taking in the next digit the programme jumps from 003. The normal input programme can clear away the remainder of the pre-input programme.

2.9 Interpreting programmes

To make programming easy an interpreting programme has been made for interpreting a simple code *). The main difficulties in programming are:

a. Scaling. To ensure that all numbers in a calculation remain in the range $1 > a \geq -1$ sometimes requires a great programming effort. To make scaling unnecessary the system of floating point operation is adopted.
b. Counting. Performing a cycle a certain number of times together with changing instructions in the cycle according to the count, is one of the most frequent components of a programme.

These two types of actions have been made extremely easy in the

*) M. V. Wilkes, D. J. Wheeler and S. Gill. The preparation of programmes for an electronic digital computer, with a special reference to the EDSAC, and the use of a library of sub-routines. Addison-Wesley Press Inc., Cambridge (Mass.) 1951.

R. A. Brooker and D. J. Wheeler. Floating operations on the EDSAC. Math. Tables Aids Comput., 7(1953)37.

R. A. Brooker. An attempt to simplify coding for the Manchester electronic computer. Brit. J. Appl. Phys., 6(1955)307.

code of which the description will be given in paragraph 2.91.

It appears that the flexibility of the ZEBRA code can be used very effectively to make the interpreting programme quick and simple. A few average times are:

Jumping and testing: 13 ms

Transport of a number to and from the store: 20 ms

Addition: 35 ms

Multiplication: 25 ms

When we assume this time to average 30 ms, and suppose the average time of an instruction in a normal programme to be 6 ms, the ratio of time between an interpreting programme and normal programming is 5 : 1 , which is very favourable. Of course standard sub-programmes can have a much better average time per instruction because optimum programming can be applied, but this makes it difficult for the programmer.

In paragraph 2.92 a few representative parts of the interpreting programme will be discussed.

2.91 The simple code

There are two stores each having 1000 locations. One store is for numbers, one store is for instructions. Addresses in the number store will be denoted by n, and addresses in the instruction store by p.

The contents of storage location n will be denoted by (n). The arithmetic unit contains an accumulator A with contents (A), and an auxiliary accumulator B. All numbers are in floating point representation $a.10^b$, where $0.1 < |a| \leq 1$ and $-9999 \leq b \leq +9999$.

The instruction code runs as follows:

Arithmetic instructions:

A n :	$(A) + (n) \to A$	Add
S n :	$(A) - (n) \to A$	Subtract
H n :	$(n) \to A$	Take in
T n :	$(A) \to n \quad 0 \to A$	Store and clear
U n :	$(A) \to n$	Store without clear
V n :	$(A)(n) \to A$	Multiply
N n :	$-(A)(n) \to A$	Negative multiply
D n :	$(A)/(n) \to A$	Divide
K n :	$(n) \to B$	Prepare accumulative multiply
V n :	$(A) + (B)(n) \to A$	Accumulative multiply
N n :	$(A) - (B)(n) \to A$	Only on V or N immediately following K

Control instructions:

X p :	Jump to p
E p :	Jump to p only if (A) > 0, otherwise proceed normally
Z :	Stop
Z p :	Special instructions
+ p :	Prepare count to p times. p = 0 is not allowed.
- :	Count and if not ready, return to instruction following the associated + instruction
R :	Can be attached to an instruction (except +, - and Z) thereby making the specified address relative to the count as far as it has gone (e.g. ARn = An + c)
+R :	Bring count of outer cycle in safety
-R :	Bring back the count of outer cycle

Input and output instructions:

L n :	Read a number from the tape and put it into n
P n :	Print (n) in floating form. Mantissa in 9 decimals, exponent in 4 decimals

Non-significant zeros in addresses must always be suppressed. Addresses 0 must always be deleted.

The coding of numbers on the tape can be done in fixed point form; e.g. +356 ; -0.005778 ; +357.2056 .Non-significant zeros may be suppressed. The number of significant decimals may not exceed 9.

Or the coding can be done in floating form:

+356	coded as	+.356E+3
-0.005778	coded as	-.5778E-2
+357.2056	coded as	+.3572056E+3

But the following is also permitted:

+357.2056	coded as	+3.572056E+2

Numbers can only be taken in by L-instructions.

The special Z-instructions are:

Z :	Stop
Z 1 :	$\sqrt{(A)} \rightarrow A$
Z 2 :	$\exp(A) \rightarrow A$
Z 3 :	$\ln(A) \rightarrow A$
Z 4 :	$\sin(A) \rightarrow A$
Z 5 :	$\cos(A) \rightarrow A$
Z 6 :	$\arctan(A) \rightarrow A$
Z 9 :	Carriage return, line feed
Z 10 :	$\log(A) \rightarrow A$

Z 11 : $\text{arccos}(A) \rightarrow A$

Z 12 : $\sinh(A) \rightarrow A$

Z 13 : $\cosh(A) \rightarrow A$

Z 14 : $\text{arcosh}(A) \rightarrow A$

Z 15 : $\text{artanh}(A) \rightarrow A$

Z 16 : $10.(A) \rightarrow A$

Z 17 : $1/10.(A) \rightarrow A$

Input of instructions is done by feeding in a tape on which the instructions to be executed are punched. This tape must be preceded by an input indication Y p which signifies: begin to put in from p onward. p can be zero; then the tape just begins with Y. On the tape the same code as for normal programmes is used with H = B and S = C .

At the end of a programme tape a second input indication must follow, giving the location of the first instruction to be executed. This has the form YpY : begin programme on p. More precisely: Y followed by blank tape = zero means: begin to execute on address mentioned in the last Yp. Thus the tape for a programme beginning in O just begins and ends with Y.

For more advanced users a number of more difficult facilities is available. Most of these facilities are connected with counting.

A large number of variants of the instructions is derived by beginning the address with a zero (normally suppressed). These orders have a different meaning. Also a O and an R can be attached together. It is irrelevant whether the O precedes the R or not. R can also be attached at the end of the address.

In having automatically variable addresses with R it is most desirable to be able to advance the address not only by unity but by an arbitrary amount. This can be effected by preparing the count with an instruction +pq and counting it off every time with q by attaching an address q - 1 to the related - instruction. In accordance with this, counting off by q = 1 is done with - with a suppressed address q - 1 = O.

Sometimes it is useful to vary an address not by adding a count but by subtracting a count, resulting in an automatic variable address running backward. This can be done by adding a O together with the R on the instructions A, S, H, V, N, D, T, U, K, X, E, V, and P. On ER only a forward running count is possible.

Counts can be preset to a calculated amount of times by using:

+On : Prepare count to the number of times, mentioned in n.

Of course this number can be O. In this case the control does not execute the next instruction but the instruction following the next instruction. In that case a jump supplied by the programmer can completely delete the cycle. E.g.:

100	+05	$(5) \geq 0$
101	X103	If $(5) > 0$ execute process normally
102	X121	If $(5) = 0$ skip process
101→103	....	
⋮		Process
120	-	
102→121	etc	

(n) are not allowed to be negative. In such a case the machine stops and gives an indication of the nature of the fault.

A possibility to modify instructions is through the instruction:

-On : Put (n) into the count register as modifier

A following R-instruction is then augmented by (n) (may be ≥ 0 or < 0). Use is made of the count register so that an outer count must be first brought in safety with +R.

It is appropriate to make some remarks about the +R and -R instructions. There are four registers for retaining inner counts. Suppose they are numbered $k = 0(1)3$. Then the action of +R and -R is:

+R : Count $\rightarrow$ k. Advance $k \rightarrow k' \equiv k + 1 \pmod 4$

-R : $k \rightarrow$ count. Set back $k \rightarrow k' \equiv k - 1 \pmod 4$

So these four registers are used cyclically in forward direction on storing a count and are used in backward direction for bringing back a previous count. By giving a number of -R instructions, it is thus possible to bring back an arbitrary count.

The significance of O on an E-instruction is as follows:

Ep : Jump to p if $(A) > 0$ otherwise proceed normally

EOp : Jump to p if $(A) < 0$ otherwise proceed normally

ERp : Jump to p+count if $(A) > 0$, otherwise proceed normally

EORp : Jump to p+count if $(A) < 0$, otherwise proceed normally

For reading in large quantities of consecutive numbers a serial read instruction has been provided:

LOn : Read numbers from the tape and put them into n and onward until a symbol Y is encountered on the tape

To facilitate the use of sub-programmes the XO instruction has been made as follows:

XOp : Store a return instruction, giving the location where the last jump came from, in p

The use of this instruction in the sub-programme is:

Main programme:

49	...	
50	X100	Jump to sub-programme beginning in 100
120 → 51	etc.	

Sub-programme:

50 → 100	X0120	Store return instruction in 120 (In this case X51)
(120	Z	Z is replaced by return instruction

The time of all types of instructions is 30 ms on the average. Relative instructions take 5 ms extra.

In a certain respect optimum programming is still possible in the simple code. A reading instruction (e.g. H, A, V, K, D, etc.) in location p can reach number location p + 1 + 8k without a waiting time. This makes the average time 5 ms shorter. A writing instruction (U and T) in location p can reach p - 1 + 8k without a waiting time. Thus the following programme takes 60 ms instead of 75 ms.

100	H5	Take number from 5. $5 \equiv 101 \pmod 8$
101	A6	Augment it by (6)
102	U5	Put it back in 5

2.92 <u>The real action of the interpreting programme</u>

Just as all numbers are written in two locations, also the instructions are split up into two parts: an address part and an operation part. The part of the store containing the simple instructions and the floating numbers begins in the real address a. The instruction pairs use a and a + 2, a + 4 and a + 6, etc. The numbers use a + 1 and a + 3, a + 5 and a + 7, etc.

For a + 1 will be written: a + 1 = b . Then an instruction address p in a simple instruction is in reality address a + 4p, and a number address n in a simple instruction is in reality address b + 4n. The abbreviations a + 4p = p' and b + 4n = n' will be used.

The use of the short registers during interpretation is:

9	count register	
10	mantissa	B-accumulator
11	exponent	B-accumulator
12	mantissa	A-accumulator
13	exponent	A-accumulator
14	extraction instruction of the form ACp'E	

15	XKB002 - XE000	normally

The form of a few representative instructions for administrative functions are:

Hm	:	X108E12 - XK4BD ACm'E4	Um	:	ADm'C13 X122QBC14
Xp	:	ACp'E X113QBC14	+p	:	A4p X100I23
Ep	:	ACp'E X117BC2	-q	:	A4q+4 X131I23

The actual interpreting programme for these instructions runs as follows:

+ ⟶	100	NE9	Pre-instruction
	101	A102BC14	Augment extraction instr. by 4
	"102	+4	
	103	ACD104	Store count limit
	(104		in 104. Clear 9
	105	ABD106E14	Store return instruction
	(106		in 106 and 14
	107	X3K4BD	Extract next instruction
H ⟶	108	A109BCE13	Tail ⟶ 13
	"109	+4	4 ⟶ B
	110	NB14	Advance extraction instruction
	111	NBE14	
	112	X3K4BD	Extract next instruction
X ⟶	113	NQE14	Store new extraction instruction
	114	NBE7	Store return instruction + 2 in 7
	115	NBC14	Extract next instruction
	116	X3K4BD	
E ⟶	117	NC12	Take mantissa
	118	A119BC14V1	Test
	"119	+4	Advance extraction instruction
	120	NBE14	If pos.: new extr. instr. If neg.: pro-
	121	X3K4BD	ceed. Extract next instruction
U ⟶	122	NQE6	Pre-instruction. Extr. instr. + 2 ⟶ B
	123	A124E4	ADn'C13 ⟶ 4
	"124	XK002-X000.7	
	125	A126CE5	ADn'+2K6 ⟶ 5
	"126	X3K5BD-XK5	X3K5BD-XK5 ⟶ 6. Extr. instr. + 4 ⟶ B.

127	NBE14	Re-store extraction instruction
128	NC12	Take mantissa
129	X4K1	Jump to 4 and execute one time
130		
→ 131	N9	Augment count
132	N	
133	A102BC14	Augment extraction instruction
134		
135	AI104E9	Re-store count, subtract test limit
136		
137	AB0106V1	If neg.:take return instr. If pos.:pro-
138		ceed
139	NBE14	Store extraction instruction
140	X3K4BD	Extract next instruction

Of course the parts for addition and multiplication are much longer. The action time for these parts is not more than really needed for the arithmetic, because these programmes are completely optimum.

The action diagram for the following short programme will be given below.

p	Xq	Jump to q
q	Um	Store in m
q+1	Hn	Take in (n)

The interpretation of the preceding instruction ended with

	NBC14	Take in extraction instr. (14) = ACp'E
	X3K4BD	Extract next instruction

The action diagram is:

A	B	C	D	
		NBC14		
	ACp'E	X3K4BD		
	ABCp'+2K	ACp'E	X3K3BD	*
ACq'E		X3K3BD		
		ABCp'+2K	X3K2BD	
	X113QBC14	X3K2BD		
		X113QBC14		*
	ACp'+1E	X114QE14		

A	B	C	D	
(ACq'E	ACp'+1E	X114QE14)		
	ACp'+2E	X115BE7		ACq'E → 14
		X116BC14		ACp'+2E → 7
	ACq'E	X3K4BD		
	ABCq'+2K	ACq'E	X3K3BD	*
ADm'C13		X3K3BD		
		ABCq'+2K	X3K2BD	
	X122QBC14	X3K2BD		
		X122QBC14		*
	ACq'+1E	X123QE6		
	ACq'+2E	A124E4	X125QE6	
ADm'+2K6C		X125QE6		ADm'C13 → 4
	ACq'+3E	A126CE5	X127QE6	Q still active
X3K5BD–XK5		X127QE6		ADm'+2K6C → 5
	ACq'+4E	X128BE14		X3K5BD–XK5 → 6
		X129C12		ACq'+4E → 14
x		X4K1		
		ADm'C13	X5K	*
y		X5K		x → m'
		ADm'+2K6C	X5K+002	
		X3K4BD		y → m'+2
	ABCq'+5K	ACq'+4E	X3K3BD	*
X108E12–XK4BD		X3K3BD		
		ABCq'+5K	X3K2BD	
	ACn'E4	X3K2BD		
	ABCn'+2K4	ACn'E4	X3K1BD	*
(n')		X3K1BD		X108E12–XK4BD → 4
		ABCn'+2K4	X4KBD	
	(n'+2)	X108E12		*
		A109BCE13	X110E12	(n') → 12
	+4	X110E12		(n'+2) → 13
		X111B14		
	ACq'+8E	X112BE14		
		X3K4BD		ACq'+8E → 14

For a total number of 39 instructions, only 17 instructions are actually specified. The points where the machine must wait are denoted by asterisks. When all these points are counted for 5 ms, then the total time of this example is 53 ms. By optimum programming in the simple code another 10 ms can be eliminated.

Especially noteworthy is the instruction X4K1 in the U-part. From this point three word pairs are stored or extracted without reference to a programme on the drum, thus eliminating waiting time for instructions.

Part 3. Simplification in the structure of machines

3.1 The essential types of operations

In the previous parts it appeared several times that by no means all the kinds of operations in a computer are indispensable. We shall now proceed to an investigation with regard to the operations that are essential. This yields the following results.

Shifting to the left in the binary system is merely doubling, this is adding to itself. So when there is an instruction to add, shifting to the left is superfluous. It is also possible to shift by multiplying; in that case the tail of the product must be kept.

Shifting to the right can be carried out by means of a multiplication by $\frac{1}{2}$, the head of the product being kept.

The stop order is superfluous because it is possible to make a conditional loop-stop as is actually done in ZEBRA.

Input and output instructions will not be considered further, because a special register can be designed in such a manner that, when a number is written in that register, this number is taken to an output unit, while another register can in a similar manner provide for the input. This procedure has already been described for the machine ZERO. Input and output also can be performed by completely separate units loading and unloading the store.

The conjunction instruction can be replaced by a cyclical programme containing shifts and test instructions. The corresponding digits of the two numbers which are to be conjugated, are consecutively shifted to the sign place and tested, after which the conjunction result is shifted into another register.

The only elements of a multiplication are testing, shifting to the left, and adding, so that a multiplication can be programmed entirely in additions and test orders. An elaborate example is given in the description of ZERO. A division can be programmed in a similar manner.

One of the orders to store is also superfluous. If only storing with clearing is present and the number in A must be kept, it can be extracted again after the storing process. Storing without clearing is also sufficient. Then clearing can be performed by subtracting the number that has just been stored.

Adding and subtracting with clearing can be cancelled, because the accumulator can be cleared with store and clear or with store without clear and subtract.

So there are left the operations store with clear (T), add (A), subtract (S), jump (X), and some form of test order. For the sake of a clear understanding the latter will not be considered as a functional digit but as an independent instruction. Sometimes a disguised operation can function as a test, just as in the ZERO, but nevertheless the hardware required for it must be provided.

One of the orders adding or subtracting is superfluous. It is quite clear that subtracting is superfluous, because $+a = -(-a)$. A sum $a + b$ can be formed by taking $-(-a - b)$.

It is not self-evident that it is also possible to maintain the addition and to omit the subtraction. $-a$ can be written as:

$$-a = -1.a = 1111111.a = a \sum 2^j = \sum a.2^j$$

Each of these separate terms can be formed by shifting to the left. Further there are only additions. It is at any rate easier to maintain the subtraction.

It is a widespread opinion that automatic computers are universal because they have a facility to discriminate, a test order. That this test order too is superfluous can be proved in the following manner. For this purpose it is most practical to put the address part of an instruction entirely to the left in the word. Let the storage capacity be 2^n, then the address digit on the utmost left hand side has the value 2^{n-1}. The programme example also makes use of addition but this can of course be programmed entirely in terms of subtractions. Then the proof can be divided into two stages:

1. Showing that it is possible to separate the leftmost digit (i.e. the sign digit) from the other digits of a number.
2. Making a variable instruction from the leftmost address digit.

The easiest way is to start with 2. When the sign digit of a number has somehow been separated from this number and put into the accumulator, a bifurcation can be made by means of the next programme:

a	A a+3	Add the constant jump instruction to the sign digit.
a+1	S a+2	Put the jump instruction in a + 2
(a+2		Jump to a + 4 or to a + 4 + 2^{n-1}
"a+3	X a+4	Constant

On $a + 4 + 2^{n-1}$ there can be an instruction to jump to a suitable location.

By means of a piece of programme of this kind point 1 can be proved. It is possible to shift the number n places to the left so that the rightmost digit is coming on the sign digit place. It is possible to test by means of the said programme whether this digit is 0 or 1 and to remove this right-hand digit from the original number. In the same manner the (n-1)th digit can be removed from the number by shifting n-1 places by testing. Finally only the sign digit is left, after which the actual test can be performed.

In this stage we should like to draw the attention to the relation existing between the three operations: test, shift to the right, and conjunction. As soon as only one of these orders is present, it is no longer difficult to execute the two other orders. Multiplication and also shifting to the right can be programmed by means of a test and an addition. If only shifting can be performed, it is easily possible to bring the sign digit in the least significant place and then a variable jump can be made. It is also possible to cut the sign digit from a number by means of conjunction only. The property, which these three operations have in common, is the fact that they all destroy a great part of the information in a word. In the adding process it is more difficult to lose information. Only on overflow information gets lost (one digit at a time and on the wrong side of the number). It seems that this loss of information is an essential element in an automatic computer. (Cf. 3.3)

The remaining operations are X, S and T. Of these operations S and T can be combined to one single order, which will be called B.

The action of B will be:

B n : $(A) - (n) \rightarrow A \qquad (A)' \rightarrow n$

So a subtraction is made, and the result is stored at the same time. That nevertheless all instructions can be performed by means of this order can be proved by showing that the S and T operations can be programmed in terms of B. However, before doing so we shall give a method to clear the accumulator with B.

a	Bn	Subtract (n). Let the result be x. Put x into n
a+1	Bn	Subtract x from x. A and n are both cleared

Now the S-operation can be coded as follows:

At the outset (A) = x, (4) = y, and (0) = 0

100	B0	$x \rightarrow 0$
101	B4	$x - y \rightarrow A \rightarrow 4$
102	B0	$- y \rightarrow A \rightarrow 0$
103	B1	
104	B1	$0 \rightarrow A \qquad 0 \rightarrow 1$
105	B4	$y - x \rightarrow A \rightarrow 4$
106	B1	$y - x \rightarrow A \rightarrow 1$
107	B4	$0 \rightarrow A \rightarrow 4$
108	B0	$y \rightarrow A \rightarrow 0$
109	B4	$y \rightarrow A \rightarrow 4$
110	B0	$0 \rightarrow A \rightarrow 0$
111	B1	$x - y \rightarrow A \rightarrow 1$

So after the operation $(A) = x - y$, $(4) = y$, and $(0) = 0$, just as required for an S-operation.

The T-operation is performed as follows:
At the outset $(A) = x$, $(4) = ?$, and $(0) = 0$.

100	B0	$x \to 0$
101	B4	
102	B4	$0 \to A \to 4$
103	B0	$-x \to A \to 0$
104	B1	
105	B1	$0 \to A$
106	B0	$x \to A \to 0$
107	B4	$x \to A \to 4$
108	B0	$0 \to A \to 0$

So after the operation $(A) = 0$, $(4) = x$, and $(0) = 0$

Thus it has been shown that every programme that can be written in terms of S and T orders can also be written in terms of B orders. Mostly, however, a problem can better be programmed directly in terms of B. For example: $(2) + (3) \longrightarrow 4$, $(2) = a$ and $(3) = b$. (2) and (3) may not be destroyed. Then the programme can be as follows:

100	B4	
101	B4	$0 \to A \to 4$
102	B2	$-a \to A \to 2$
103	B3	$-a - b \to A \to 3$
104	B4	$-a - b \to 4$
105	B3	$0 \to A \to 3$
106	B2	$a \to A \to 2$
107	B0	$a \to A \to 0$
108	B3	
109	B3	$0 \to A \to 3$
110	B4	$a + b \to A \to 4$
111	B0	$b \to A \to 0$
112	B3	$b \to A \to 3$
113	B0	$0 \to A \to 0$

The elucidation is self-explanatory.

Now the machine knows only two types of instructions: X and B. As a last step we shall discuss the cancelling of the X-operation. Inside the machine the alternation between instruction period and operation period must still be maintained.

This can be effected by an automatic alternator. The contents of the control register are then alternately used for the B-operation and for the intermediate X-operation. An external jump can be made by interrupting the normal alternation, just as this has been done in all other projects of this thesis. For this interruption the address 0 can be used very effectively.

The operational part will then be:

X n : $(C) \rightarrow D \quad (n) \rightarrow C$ The next instruction is of the type B, if $(D) \neq 0$, otherwise it is again an instruction of the type X

B n : $(D) + 1 \rightarrow C \quad (C) \rightarrow D \quad (A) - (n) \rightarrow n \quad (A) - (n) \rightarrow A$
The next instruction is of the type X

A zero detector tests whether the number that flows out of B is 0. In this case the normal alternation $X \longleftrightarrow B$ is interrupted. This zero detector is a device which can be constructed very easily in a serial machine. It consists of a storing element which is set on the first 1 flowing out of B. If, however the number does not contain 1 then at the end of the number the device is still in the rest condition. After having taken into account the position of this zero detector it can be reset again to zero.

An example of a programme is:

2	10	B-operation with address 10
3	12	B-operation with address 12
4	0	Prepare jump
5	8	Jump to 8
8	13	B-operation with address 13

The action in the control is:

C	D	
2		X
10	2	B
3	10	X
12	3	B
4	12	X
0	4	B
5	0	X. Here (D) = 0. So the following operation is again X.
8	5	X
13	8	
etc.		

3.2 The purely one-operation machine

Though in the preceding paragraph the operation part of an instruction could entirely be deleted, a disguised jump order was nevertheless maintained, because a special sequence of addresses was used.

In this paragraph it will be shown that even this is not essential but that by means of a purely jumpless machine all the operations can yet be carried out. From a technical point of view this only means that the zero detector on B is no longer needed. Now the X - B alternator is always operated. So internally there still exists: extract a new instruction, but externally the jump does no longer exist. The consequence is that the machine is only able to run through all the instructions in the store sequentially.

Operation digits are no longer required. It will be assumed that the address fills the entire word of n bits and that the store contains 2^n registers. It will have to be shown that, by means of a certain fixed programme, part of the store can be used for a programme in a normal code that is to be interpreted.

We may suppose all the normal orders such as add, subtract, multiply, test, jump, etc. to occur in the code to be interpreted.

As a jump dependent on a number can no longer be made, but as it is still possible to extract a number from a variable address calculated by the machine, an operation can be interpreted as follows:

First determine the results of all the operations that are possible and then select the correct answer. By answers are not only meant the arithmetic answers but also the words that have to go to the registers for simulation of the control registers belonging to the machine to be interpreted.

Now the problem can be split up into two parts:

1. Performing the actual operations.
2. Running through all the words in the remainder of the store (including the programme to be interpreted).

Numbers may not be run through, because otherwise these arbitrary numbers are carried out as instruction. These instructions can be quite harmful. That is why these numbers must be put in safety by means of a transfer programme. The programme meant under 1 will be called the active programme.

In the first place it must be proved that all the operations required in the code to be interpreted, can be carried out. Strictly speaking it will be sufficient, if it is proved that a B-instruction and a jump can be interpreted. Multiplication etc. can, however, also be interpreted directly.

In the second place it must be proved that it is possible to make the transfer part.

Interpreting B and X comes down to the following activities:

B n :	$(C) + 1 \rightarrow C$	$(A) - (n) \rightarrow n$	$(A) - (n) \rightarrow A$
X n :	$n \rightarrow C$	$(n) \rightarrow n$	$(A) \rightarrow A$

In all cases $(C) + 1$, and $(A) - (n)$ are calculated by the active part. This is simply possible with a stretched programme. These intermediate results can be put together with n, (A), and (n) into working registers. Let us suppose that the leftmost 1 in a word in the code to be interpreted indicates whether an operation is a B or an X. This digit can be separated in accordance with a method which is analogous to the one mentioned in the preceding paragraph and applied to avoid the test operation. With this method it is not possible to make a variable jump, but the extracted digit can be used to form a variable address.

The complete process is then as follows:
Shift the number (instruction to be interpreted) so far to the left that the rightmost digit comes entirely at the left. Make a variable instruction from it. Extract by means of it the constant 0 or 1 from location x or from the location which is situated diametrically with respect to x. Subtract this from the original number. Remove in the same manner also the other digits from the number except the leftmost one.

Thus it is possible te extract the address n from the instruction, to extract (n) and to make a variable storing order to n.

By means of the leftmost digit of the instruction to be interpreted (so the operation part) three variable orders can be made which bring:

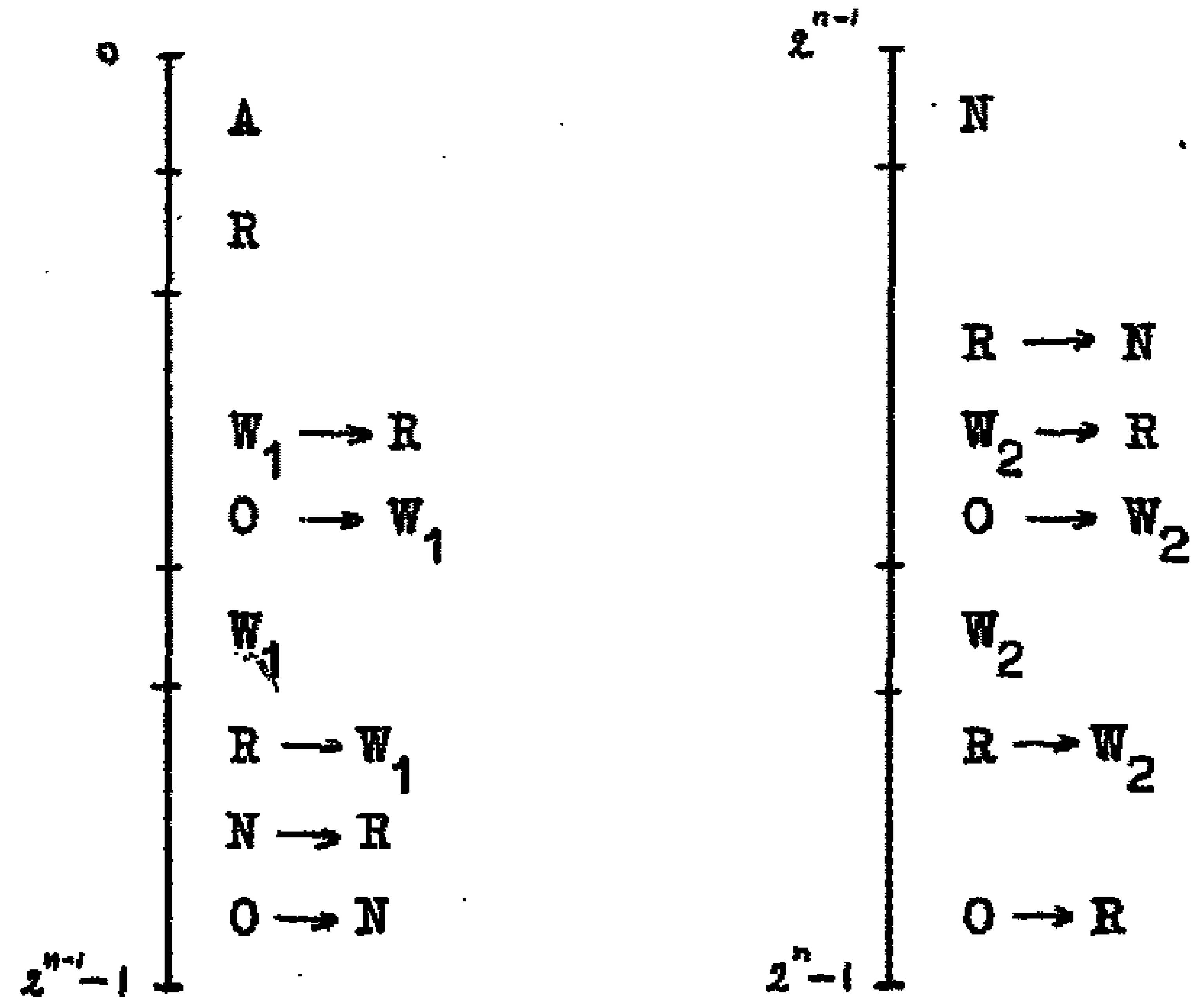
$(x) = (C) + 1 \rightarrow C$		$(x + 2^{n-1}) = n \rightarrow C$
$(y) = (A) - (n) \rightarrow n$	or	$(y + 2^{n-1}) = (n) \rightarrow n$
$(z) = (A) - (n) \rightarrow A$		$(z + 2^{n-1}) = (A) \rightarrow A$

Now the interpretation has been completed.

When programmed out in terms of B and X, several operations require a number of B and X-operations of the order of n^2 (such as test) or n^3 (such as multiplication). In any case the number of instructions required can be contained in the store when only n is greater than a certain amount N, because 2^n goes up faster than n^3. It is extremely difficult to make an exact quantitative statement, because this depends very much upon the exact way of constructing the interpreting programme. As a rough estimate N will be 25.

The transfer programme starts from the structure of the active programme plus the working registers. Let us suppose that this part consists of four parts of equal size: the active programme (A), half of the working registers and the constants (W_1), the diametrical half of the working registers and finally the effective part (N), which can contain the programme to be interpreted. These four parts can always be made of the same size by filling them with zeros. (A register containing 0 with (A) = 0 is a dummy instruction, because $0 \rightarrow A \rightarrow n$

Furthermore the transfer programme requires a group of spare registers R in which A, N, W_1, or W_2 can be stored temporarily. The structure of the entire programme can be as follows:



At the outset R is entirely cleared. Then A is carried out. R is traversed without damage, W_1 is put in safety, cleared, and traversed, etc. All these transfer parts are stretched programmes, which require quite a number of instructions for each word to be transferred. So by indicating a possibility of solving the problem the proof has been completed. The given solution need by no means be the best.

3.3 Conclusion

In "On computable numbers", TURING *) does not exercise restraint as regards the size of the machine or the extent of the store. Therefore the class of the computable numbers is infinite. From a practical point of view it is, however, better to restrict oneself to finite machines. Then the latter can no longer be called universal in the sense of TURING, because they cannot generate all "computable numbers".

As it has already appeared from the designs in this thesis arithmetic unit and control cannot be rigorously separated. Together they may be called the operational part of the machine. On the other hand there is the store (plus the selection mechanism) which is supposed to be of uniform structure.

We shall not discuss two-level stores, nor input and output. We can imagine the machine to be filled with a separate

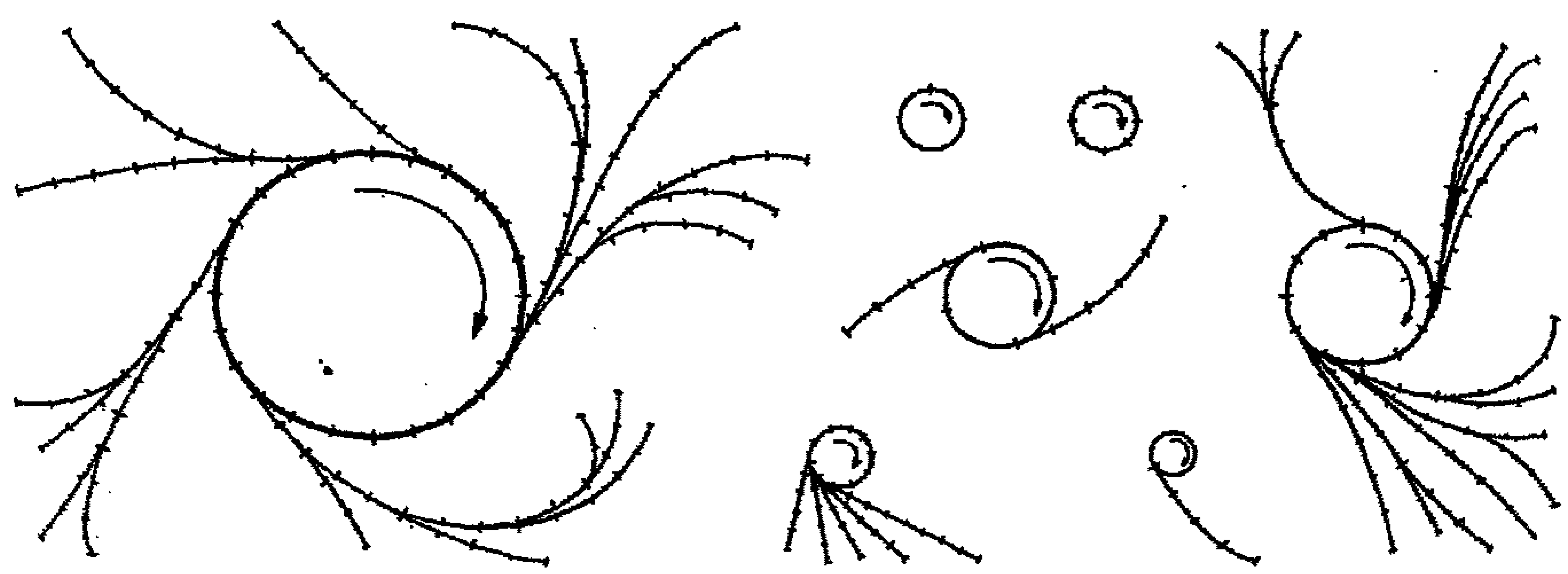
*) A. M. Turing. On computable numbers. Proc. Lond. Math. Soc. 42(1938)230.

external process. After that the machine operates and then the results can be read directly from the store. Furthermore only the addresses need play a rôle in the instructions. The action of the machine can be considered to be effectuated in elementary steps, which e.g. consist of one word period. Parallel and serial machines can be dealt with from a uniform point of view.

It is possible to define a number which represents the state of the whole machine at a certain moment. The state of the store can be represented by making the contents of all the registers together form one long number. To these contents must be added the contents of the registers of the operational part plus the contents of some isolated storage elements for single bits. The long number obtained in this manner, determines the internal state of the machine. Let e.g. the length of the number in a register be n. Then the maximum size of the store can amount to 2^n registers. Let us suppose that in the operational part there are p single word registers and k single bit storage elements. Then the number defining the state contains $n.2^n + p.n + k$ bits.

As the machine functions on a strictly causal basis, a certain state will also determine uniquely the next state, etc. So at a given initial condition and at a given structure of the machine the course of the computation is altogether fixed. As the machine is supposed to be finite and as there is consequently only a finite number of states, a sequence of states must finally end in a cycle with a certain period, because, as soon as the same state is reached again the next state is the same again as before.

It is not necessary that the initial point too should lie on the cycle, it can also lie on an "offshoot". There can (and mostly will) be more than one cycle. So the state diagram in principle looks like this:



Among the enormously large number of initial states which are possible, there are only some which perform a function to which we attach significance.

If a problem stops at the end, this can be realised by making a loop-stop, which means that the route in the state diagram ends on a point-circle.So the actual problem lies entirely on the offshoots. A problem that lies entirely on a circle, larger than a point-circle, cannot stop. The "result" is dependent on the moment at which the state is being examined. This represents e.g. a useful problem in the case of determining a pseudo-random number (given by the contents of a certain register at an arbitrary moment). Most problems lie, however, on the offshoots.

In a computer three kinds of elements can be distinguished:
1. Conservative elements.
2. Reducing elements.
3. Dissipative elements.

In a conservative element no information is lost. All information that is going into it, also leaves it again, be it in a changed form. Examples are the delay line, and the invertor. From the result it can be deduced back what the data have been. As long as in the state diagram states are traversed which make use only of conservative elements, there are no junction points.

Reducing elements decrease the quantity of information flowing into them. The input data cannot be deduced back from the output only. Example: an adder. It is, however, possible to deduce part of the input information from the rest of the input information together with the output information. (E.g. when $a + b = c$ then $a = c - b$). So in some switching circuits an adding unit need not lose information provided that the required part of the input information is retained in another manner. It is, however, also possible to lose part of the information. So in doubling a number it is not possible to reconstruct the lost digit on the left, but all other digits can be reconstructed by shifting to the right, because we know in this case that the result has been formed by shifting to the left. So dependent on the use made of these reducing elements, they are sometimes reversible, sometimes irreversible.

The dissipative elements are at any rate irreversible. They cause a certain part of the information supplied to them, to be lost. Examples are conjunction and disjunction elements. These elements are essentially irreversible. In the state diagram the junction points are caused by dissipative elements or sometimes by reducing elements.

For some structures with state diagrams which are mainly cyclical, DUPARC *) and VAN WIJNGAARDEN **) have considered the period of the cycle.

So the fact that by far the greater number of problems are

*) H. J. A. Duparc. Divisibility properties of recurring sequences. Thesis, Amsterdam, 1953.

H. J. A. Duparc. Periodicity properties of recurring sequences. Proc. Kon. Ned. Akad. Wetensch., Amsterdam, A57(1954)331 & 473.

**) A. van Wijngaarden. Dynamica van rekenmachines. Syllabus colloquium Moderne Rekenmachines, (1954) March, 27th. Mathematisch Centrum, Amsterdam.

lying on the offshoots in the state diagram, is mainly caused by the action of the dissipative elements. The loss of information in these elements appears to play an essential rôle in computing machines. This is perfectly in accordance with the irreversible character of causal processes.

The author has not found an essential difference between an arbitrary machine with a certain state diagram with offshoots and any normal computer. At best one can say that the number of problems that can be tackled in a machine with a larger number of offshoots is larger than the number of problems that can be processed in a simple machine with a simpler state diagram. Furthermore the appreciation of what a machine can do is highly dependent upon what we call a problem, and what we consider useful transformation rules.

A problem that arises here, is: what is the minimum operational part required to control a store of a certain size effectively?

This problem greatly resembles the problem dealt with in paragraphs 1.63 and 2.86: what is the smallest possible pre-input programme at a given structure of the machine?

As shown in the previous paragraph, various orders are not necessary. If we do not object to the enormous slowing down, we can yet make a one-operation machine normally usable for every code desired, viz. by means of an interpreting programme. From the beforegoing it appears that some of the functions of the operational part can be transferred to the store. This interpreting programme can operate in cascade: a central programme interprets a programme that has more facilities, and this programme in its turn interprets an extensive code. This requires, however, a considerable part of the store. By means of an operational part with a word-length of n digits a store with only 2^n registers can be covered. So there will be a value n, with which it is no longer possible to get the desired interpreting programme into the store. Then we are faced by the situation that the "catalogue fills up the entire library". If the operational part is made more complicated, the lower bound of the capacity of the store can be brought down. It is more a question of economy to determine the optimum capacity of the store, and the complication of the operational part, in relation to the speed and the price.

It is very remarkable that the machine ZERO is already a practicable machine, though it is hardly more complicated than the one-operation machine, which is completely unpractical. Perhaps a certain optimum has been found already intuitively.

The problem of the simplest pre-input programme shows a certain conformity in so far that there seems to be a minimum simplicity, which for PTERA amounts to 5 instructions. By means of these instructions a certain length of tape can be controlled effectively in the same manner as it happens with a cascaded interpreting programme. In various stages the normal input programme is put in, which programme in its turn can organise again larger tapes (or parts of the store). So if the machine only has at its disposal the pre-input programme and a certain length of tape (here the tape entirely plays the rôle of store) on which all the data required for a normal input programme have been put, then no more useful problems can be tackled, because the entire store has been filled with the necessary data.

In this last problem the qualitative character of these considerations are clearly shown. For the time being it still depends on the inventiveness of the programmer or engineer whether the limit of simplicity can be economically reduced a little. A need for a more exact calculus for treating these problems is still needed.

References

E. M. Bradburd.
Magnetostrictive delay lines.
Electrical Communication, 28(1951)46.

R. A. Brooker.
An attempt to simplify coding for the Manchester electronic computer.
British Journal of Applied Physics, 6(1955)307.

R. A. Brooker and D. J. Wheeler.
Floating operations on the EDSAC.
Mathematical Tables and other Aids to Computation. 7(1953)37.

A. W. Burks, H. H. Goldstine and John von Neumann.
Preliminary discussion of the logical design of an electronic computing instrument, 2nd edition.
The Institute for Advanced Study, Princeton, N. J. 1947.

H. J. A. Duparc.
Divisibility properties of recurring sequences.
Thesis, Amsterdam, 1953.

H. J. A. Duparc.
Periodicity properties of recurring sequences.
Proceedings Koninklijke Nederlandse Akademie van Wetenschappen, Amsterdam, A57(1954)331 & 473.

A. L. Freedman.
Elimination of waiting time in automatic computers with delay-type stores.
Proceedings Cambridge Philosophical Society, 50(1954)426

W. L. van der Poel.
A simple electronic digital computer.
Applied Scientific Research, B2(1952)367.

W. L. van der Poel.
Dead programmes for a magnetic drum automatic computer.
Applied Scientific Research, B3(1953)190.

W. L. van der Poel.
Het programmeren voor PTERA.
Het PTT-Bedrijf, 5(1953)135.

W. L. van der Poel.
De werking van PTERA.
Het PTT-Bedrijf, 5(1953)124.

Staff of the Computation Laboratory.
Description of a magnetic drum calculator.

The Annals of the Computation Laboratory of Harvard University, Vol.25. Harvard University Press, Cambridge (Mass.) 1952.

A. M. Turing.
On computable numbers.
Proceedings London Mathematical Society, 42(1938)230.

M. V. Wilkes.
The use of a "floating address" system for orders in an automatic digital computer.
Proceedings Cambridge Philosophical Society, 49(1953)84.

M. V. Wilkes and J. B. Stringer.
Micro-programming and the design of the control circuits in an electronic digital computer.
Proceedings Cambridge Philosophical Society, 49(1953)230.

M. V. Wilkes, D. J. Wheeler and S. Gill.
The preparation of programmes for an electronic digital computer, with a special reference to the EDSAC, and the use of a library of sub-routines.
Addison-Wesley Press Inc., Cambridge (Mass.) 1951.

D. J. Wheeler.
Programme organization and initial orders for the EDSAC.
Proceedings of the Royal Society, A202(1950)573.

A. van Wijngaarden.
Dynamica van rekenmachines.
Syllabus Colloquium Moderne Rekenmachines, (1954) March, 27th. Mathematisch Centrum, Amsterdam.

E R R A T A

page	line	
16	24	$\|(A)\|$ becomes $\|(R)\|$
16	28	$\|(A)\|$ becomes $\|(R)\|$
17	0	Insert heading: Output
76	30	12 becomes 10
77	6	recording becomes recoding
93	23	ABCq'+5K becomes ABCq'+6K
93	25	ABCq'+5K becomes ABCq'+6K .

S T E L L I N G E N

0.

Een inrichting voor het automatisch verkrijgen van de juiste terugkeerinstructie bij het gebruik van gesloten subprograms in een rekenmachine heeft voor het gemakkelijk programmeren veel waarde en is op elk type machine eenvoudig te maken.

1.

Het is in een rekenmachine van voordeel om een accumulator van dubbele lengte (resp. twee accumulatoren van enkele lengte die gemakkelijk gekoppeld kunnen worden) ter beschikking te hebben.

2.

Het is aanbevelenswaardig om in informatieverwerkende machines slechts enkele zorgvuldig voorbereide en beproefde standaardschakelingen, uitgevoerd als insteekbare eenheden, te gebruiken. Daarbij behoort van zo min mogelijk buistypen en voedingsspanningen gebruik gemaakt te worden.

3.

Het invoeren van een genormaliseerd gebruik van benamingen voor de meest voorkomende electronische schakelelementen is zeer gewenst.

4.

De bewering van WILKES dat een systeem van drijvend adresseren, waarbij alleen naar reeds ingezette adressen verwezen kan worden, van geen of weinig waarde zou zijn, is onjuist.

M. V. Wilkes. The use of a "floating address" system for orders in an automatic digital computer.
Proc. Camb. Phil. Soc., 49(1953)84.

5.

LUBKIN heeft trachten aan te tonen dat een invoerorgaan dat niet onderling gesynchroniseerd is met de interne machine nooit met volledige zekerheid (waarbij de electronische elementen als ideaal beschouwd worden) informatie kan invoeren. Deze bewering houdt echter geen steek.

S. Lubkin. Asynchronous signals in digital computers.
Math. Tables Aids Comput., 6(1952)238.

6.

Een inzetaanwijzing voor het als parameter vastleggen van het adres volgend op het laatste adres van een subprogram als eerste adres van een volgend subprogram kan beter aan het eind van een subprogram dan aan het begin van het volgend subprogram gegeven worden.

7.

Als $v_0 = 2$, $v_1 = 1$ en $v_k = v_{k-1} + v_{k-2}$ voor $k \geq 2$, dan volgt uit het priem zijn van p, dat $v_p \equiv 1 \pmod p$. Ondanks het feit dat omgekeerd uit $v_n \equiv 1 \pmod n$ niet de primaliteit van n volgt, kan deze stelling toch van groot nut zijn bij het onderzoek naar de primaliteit van getallen.

8.

De manier waarop OBERMAN enige nieuwe logische verbindingstekens in de schakelalgebra invoert is aanvechtbaar. De wijze waarop hij enige gelijkheden betreffende de door hem ingevoerde aftrekking tracht te staven, is niet gefundeerd.

R. M. M. Oberman. De bewerkingstekens in de schakelalgebra. Het PTT-Bedrijf, 6(1954)1.

9.

Men kan de optische doorrekeningsformules door het invoeren van geschikte variabelen zodanig omvormen dat er een bijna volkomen dualiteit ontstaat tussen de grootheden en formules die de overgang van oppervlak tot oppervlak beschrijven en de grootheden en formules die de breking aan een oppervlak beschrijven. Deze dualiteit strekt zich verder uit dan het reeds door HERZBERGER uitgesproken beginsel.

M. Herzberger. Über ein Dualitätsprinzip in der Optik. Zeitschr. für Physik, 91(1934)323.

10.

Bij het onderwijs in het pianospel is het laten spelen van toonladders met uniforme vingerzetting van groot nut.

11.

Bij blinden zijn de andere zintuigen niet hoger ontwikkeld dan bij zienden. Zij hebben er alleen beter gebruik van leren maken. Bijv. kunnen zij op grond van bepaalde verworvenheden een bijzondere "aanleg" voor programmeren bezitten.

12.

Het is waarschijnlijk, dat de terugkoppeling nodig voor het beheersen van de spraak slechts voor een deel via het gehoor en voornamelijk via het gevoel gaat.

Behoort bij academisch proefschrift: W.L.v.d.Poel,
The logical principles of some simple computers.

S A M E N V A T T I N G

In dit proefschrift worden enige ontwerpen van eenvoudige rekenmachines behandeld. In deel 1 wordt een reeds voltooide rekenmachine, genaamd PTERA besproken. In deel 2 wordt een nieuw ontwerp genaamd ZEBRA behandeld. Deze beide ontwerpen, benevens een reeds eerder door de schrijver gepubliceerd ontwerp gaven aanleiding tot enkele theoretische beschouwingen die in deel 3 zijn verenigd.

Alle ontwerpen maken ter wille van de eenvoud gebruik van het tweetallige stelsel. De getallen worden steeds in serievorm getransporteerd en het geheugen is steeds een magnetische trommel. Deze vorm van geheugen is gekozen op grond van zijn betrouwbaarheid en lage prijs, hoewel de behandelde principes evengoed toepasbaar zijn op andere vormen van seriegeheugens.

De PTERA is een machine waarbij is uitgegaan van enkele bestaande schakelingen voor het rekenorgaan en voor het geheugen. De bijzonderheden zijn gelegen in de wijze van organisatie van de besturingsregisters. Gedurende het halen van een volgende instructie verricht het rekenorgaan in de meeste andere machines geen rekenhandeling. Het is dus in die tijd beschikbaar om ten behoeve van de besturing enige functies te verrichten, met name het ophogen van de instructie die de volgende instructie moet halen. Een tweede bijzonderheid is het meevoeren van een operatiegedeelte door de instructie die de volgende instructie moet halen. Dit betekent dat ook de instructiecyclus en niet alleen de operatiecyclus een nuttige handeling kan verrichten. Nadelen van de PTERA zijn de geringe snelheid en het niet volledig benutten van de mogelijkheden van de bovengenoemde beginselen. Deze bezwaren zijn in de ZEBRA ondervangen.

Tijdens de bouw van de PTERA werden naast elkaar ontwikkeld de ZERO, een machine die reeds eerder beschreven is en die gediend heeft als experimentele machine, en de ZEBRA. Naast de bijzondere wijze van behandeling van de besturing maken deze projecten gebruik van functionele cijfers in het operatiegedeelte van de instructies. Hierdoor kon in ZEBRA een grote flexibiliteit en een grote eenvoud bereikt worden. Door de aanwezigheid van de functionele cijfers kon ook de instructiecyclus in hoge mate effectief gemaakt worden. De eenvoud eiste het ontbreken van een ingebouwde vermenigvuldiger en deler. Dit gaat evenwel niet ten koste van snelheid of van kortheid van programmeren. De grote winst in snelheid werd bereikt door optimale programmering. Aan de hand van vele programs worden de mogelijkheden van dit project toegelicht; o.a. wordt de organisatietechniek van interpreteerprograms voor rekenen met drijvende komma besproken.

Deel 3 behandelt de probleemstelling: wat is in een rekenmachine essentiëel en hoe ver kan men gaan met vereenvoudiging. Als uitgangspunt werd de ZERO gekozen. Bewezen wordt dat vele soorten operaties niet ingebouwd behoeven te worden, o.a. vermenigvuldigen, geconditioneerde operaties, schuiven, enz. Door invoeren van een nieuw type operatie wordt bewezen dat slechts één soort operaties essentiëel is. Zelfs de sprongopdracht blijkt niet noodzakelijk te zijn. Besloten wordt met enige kwalitatieve beschouwingen over de tot het uiterste vereenvoudigde ontwerpen.